



Production Planning and Scheduling (PPS) Version 1.0

Committee Specification 01

29 September 2011

Specification URIs

This version:

<http://docs.oasis-open.org/pps/pps/v1.0/cs01/pps-v1.0-cs01.pdf> (Authoritative)
<http://docs.oasis-open.org/pps/pps/v1.0/cs01/pps-v1.0-cs01.html>
<http://docs.oasis-open.org/pps/pps/v1.0/cs01/pps-v1.0-cs01.doc>

Previous version:

N/A

Latest version:

<http://docs.oasis-open.org/pps/pps/v1.0/pps-v1.0.pdf> (Authoritative)
<http://docs.oasis-open.org/pps/pps/v1.0/pps-v1.0.html>
<http://docs.oasis-open.org/pps/pps/v1.0/pps-v1.0.doc>

Technical Committee:

OASIS Production Planning and Scheduling TC

Chair:

Yasuyuki Nishioka (nishioka@hosei.ac.jp), PSLX Forum / Hosei University

Editors:

Yasuyuki Nishioka (nishioka@hosei.ac.jp), PSLX Forum / Hosei University
Koichi Wada (wada@kt.rim.or.jp), PSLX Forum

Additional artifacts:

This prose specification is one component of a Work Product which also includes:

- XML schema: <http://docs.oasis-open.org/pps/pps/v1.0/cs01/xsd/pps-schema-1.0.xsd>

Related work:

This specification replaces or supersedes:

- PPS (Production Planning and Scheduling) Part 1: Core Elements, Version 1.0
- PPS (Production Planning and Scheduling) Part 2: Transaction Messages, Version 1.0
- PPS (Production Planning and Scheduling) Part 3: Profile Specifications, Version 1.0

Declared XML namespace:

<http://docs.oasis-open.org/ns/pps/2011>

Abstract:

OASIS Production Planning and Scheduling (PPS) specification deals with problems of decision-making in all manufacturing companies who want to have a sophisticated information system for production planning and scheduling. PPS specification provides XML schema and communication protocols for information exchange among manufacturing application programs in the web-services environment. The Core Elements section focuses on information model of core elements which can be used as ontology in the production planning and scheduling domain. Since the elements have been designed without particular contexts in planning and scheduling, they can be used in any specific type of messages as a building block depending on the context of application

programs. The Transaction Messages section focuses on transaction messages that represent domain information sent or received by application programs in accordance with the context of the communication, as well as transaction rules for contexts such as pushing and pulling of the information required. Finally, the Profile Specifications section focuses on profiles of application programs that may exchange the messages. Application profile and implementation profile are defined. Implementation profile shows capability of application programs in terms of services for message exchange, selecting from all exchange items defined in the application profile. The profile can be used for definition of a minimum level of implementation of application programs which are involved in a community of data exchange.

Status:

This document was last revised or approved by the OASIS Production Planning and Scheduling TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/pps/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/pps/ipr.php>).

Citation format:

When referencing this specification the following citation format should be used:

[PPS]

Production Planning and Scheduling (PPS) Version 1.0. 29 September 2011. OASIS Committee Specification 01. <http://docs.oasis-open.org/pps/pps/v1.0/cs01/pps-v1.0-cs01.html>.

Notices

Copyright © OASIS Open 2011. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS" and "PPS" are trademarks of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1	Introduction	6
1.1	Terminology	6
1.2	Normative References	7
1.3	Non-Normative References	7
1.4	Terms and definitions	7
2	Core Elements	9
2.1	Primitive Elements	9
2.1.1	Structure of primitive elements	9
2.1.2	List of primitive elements	10
2.2	Relational Elements	12
2.2.1	Structure of relational elements	12
2.2.2	List of relational elements	14
2.3	Specific Elements	15
2.3.1	Structure of specific element	15
2.3.2	List of specific elements	16
2.4	Eventual Elements	17
2.4.1	Structure of eventual element	17
2.4.2	List of eventual elements	18
2.5	Accounting Elements	18
2.5.1	Structure of Accounting element	18
2.5.2	List of accounting elements	19
2.6	Administrative Elements	20
2.6.1	Structure of Administrative Elements	20
2.6.2	List of Administrative Elements	20
2.7	Data Elements	21
2.7.1	Qty element	21
2.7.2	Char element	22
2.7.3	Time element	22
3	Transaction Messages	24
3.1	Messaging model	24
3.1.1	Basic Unit of messaging	24
3.1.2	Message classes	24
3.1.3	Messaging models	25
3.1.4	Procedures on responders	27
3.2	Add, Change and Remove (PUSH model)	28
3.2.1	Add transaction	28
3.2.2	Change transaction	29
3.2.3	Remove transaction	31
3.3	Notify and Sync (NOTIFY and SYNC model)	32
3.3.1	Notify transaction	32
3.3.2	Synchronizing process	33
3.4	Information Query (PULL model)	35
3.4.1	Target domain objects	35

3.4.2	Target domain property	37
3.4.3	Multiple property (Level 2 function)	39
3.4.4	Using Header element.....	41
3.4.5	Show document.....	42
3.5	XML Elements	43
3.5.1	Message Structure	43
3.5.2	Transaction element.....	44
3.5.3	Document element	45
3.5.4	Error element.....	47
3.5.5	App element	48
3.5.6	Condition element	48
3.5.7	Selection element.....	49
3.5.8	Header element.....	50
3.5.9	Property element	50
4	Profile Specifications	53
4.1	Application profile Definitions	53
4.1.1	General.....	53
4.1.2	Structure of profile definitions.....	53
4.1.3	Standard profile definitions.....	54
4.1.4	Extended profile definitions	55
4.1.5	Revision rule.....	56
4.2	Implementation profiles.....	56
4.2.1	General.....	56
4.2.2	Structure of implementation profiles.....	57
4.2.3	Level of implementation	59
4.2.4	Profile inquiry.....	59
4.3	XML Elements	60
4.3.1	AppProfile Element.....	60
4.3.2	AppDocument Element	60
4.3.3	AppObject Element	61
4.3.4	AppProperty Element	62
4.3.5	Enumeration Element.....	62
4.3.6	EnumElement Element.....	63
4.3.7	ImplementProfile Element	63
4.3.8	ImplementDocument Element.....	65
4.3.9	ImplementAction Element	66
4.3.10	ImplementProperty Element.....	66
4.3.11	ImplementEvent Element	67
5	Conformance	69
Appendix A.	Object Class diagram of Core Elements.....	70
Appendix B.	Cross reference of elements.....	71
Appendix C.	Implementation level	73
Appendix D.	Revision History	74
Appendix E.	Acknowledgements.....	75

1 Introduction

This specification focuses on production planning and scheduling for all kinds of products and services provided by manufacturing enterprises. Production scheduling applications dealt in this specification can be divided into scheduling in the whole enterprise including some areas and sites, and detailed scheduling within an individual area and work-centers.

The scope of this specification, however, doesn't include optimization logic for solution, special knowledge of individual enterprises, concrete solution methods for production planning and scheduling, and planning problems for the total supply chain.

Section 2 of this specification prescribes how to describe contents of the XML messages which are used for exchanging the information on Production Planning and Scheduling by some application software programs.

If information defined with PPS is exchanged between production planning and scheduling applications, the enterprise can develop systems easily at a low cost and make them more competitive for the whole enterprise. In order to do this, the systems have to have high extendability as well.

Section 3 of this specification provides structure and rules of XML transaction elements for messaging between two application programs. Main parts of XML representations of the messages consist of XML core elements defined in Section 2. Those specifications define additional XML elements and attributes that are needed to establish such communications.

From perspective of planning and scheduling in manufacturing management, there are many kinds of domain documents and domain objects. All of that information are sent or received in particular context such as notifying new information, requesting particular information, and so forth. Section 3 prescribes communication protocols by categorizing such various transactions into simple models. The specification doesn't focus on the underlying communication protocols, such as HTTP, SMTP and FTP.

A transaction element has message documents which are sent or received as a message. This part does not define type of document, but defines a data structure of message elements, transaction elements and document element that may be created for any particular circumstances. Each document element has domain objects in the production planning and scheduling domain. The domain objects can be represented by nine primitive elements defined in Section 2.

This specification also defines messaging models of communication between two application programs, where transaction elements are sent as a message. In the messaging model, an initiator can request a service such as add, change and remove information to the responder. The initiator is also able to request of getting information by sending a query-like-formatted message. This specification defines syntax and rules for such messaging models.

Section 4 of this specification prescribes definition of application profile and implementation profile. Implementation profile shows capability of information exchange with other application programs using PPS transaction messages. In order to define an implementation profile for each application program, this document also defines and prescribes application profile specification that should be consistent with all implementation profiles. An application profile allows each individual program to describe their capability.

Application profile shows a set of domain documents, domain objects and domain properties, which may be used in a message of production planning and scheduling application programs. Implementation profile shows domain documents, domain objects and domain properties that the application program can deal with correctly. The implementation profile also shows an implementation level of the application program. By collecting implementation profiles, a system integrator can arrange particular messaging in accordance with application specific scenarios.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

49 1.2 Normative References

- 50 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
51 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 52 [PCRE] PCRE(Perl Compatible Regular Expression), <http://www.pcre.org/>
- 53 [PATH] XML Path Language (XPath) Version 1.0, <http://www.w3.org/TR/xpath>

54 1.3 Non-Normative References

- 55 [PSLXWP] PSLX Consortium, PSLX White Paper - APS Conceptual definition and
56 implementation, <http://www.pslx.org/>
- 57 [PSLX001] PSLX Technical Standard, Version 2, Part 1: Enterprise Model (in Japanese),
58 Recommendation of PSLX Forum, <http://www.pslx.org/>
- 59 [PSLX002] PSLX Technical Standard, Version 2, Part 2: Activity Model (in Japanese),
60 Recommendation of PSLX Forum, <http://www.pslx.org/>
- 61 [PSLX003] PSLX Technical Standard, Version 2, Part 3: Object Model (in Japanese),
62 Recommendation of PSLX Forum, <http://www.pslx.org/>

63 1.4 Terms and definitions

64 Plan

65 Unit for intensive information of related orders corresponding to a specific period on a discrete
66 time scale, or calculated information based on the schedule under the related orders. This can
67 represent actual results when the related events have been occurred.

68 Order

69 Unit of requirement describing concrete item, resource or operation in a specific place at a
70 specific time. This can also represent the results to the requirement.

71 Party

72 Customer who is a sender of an order and has a demand to make a decision, or supplier who is a
73 receiver in case that a decision-maker sends the demand that can't be handled inside.

74 Item

75 Object to be produced or consumed by production activities. The quantity or the quality of item is
76 changed during the production activity. Examples include product, parts, module, unit, work in
77 process and materials.

78 Resource

79 Object that can provide essential function for production activities. The capacity of function is
80 used during production activity, and is available again after finishing the production. Examples
81 include equipment, machine, device, labor and tool.

82 Process

83 Segment of production activities indicating a certain production line or method. This takes
84 duration from start time to end time, and gives added value to the producing item. One process
85 may have two or more than two processes detailed in the lower levels.

86 Lot

87 Instance of a specific volume of item that exists in a specific place at a specific time. Generally
88 the specific time corresponds to start or end of an operation, and the specific volume is equal to
89 the quantity of item produced or consumed by the operation.

90 Task

91 Unit of necessity to execute a specific operation at a specific time, indicating the volume of used
92 capability provided by the applicable resource. This can represent both capacity value provided

93 by resource at a specific time point, and aggregated total value of capacity provided by resource
94 during specific duration.

95 **Operation**

96 Actual processing element to be executed by a specific task, and to produce or consume a
97 specific lot. It is a concrete instance of particular processes in production activities.

98 **Application profile**

99 Collections of profile specifications for all application programs that may be involved in the
100 communication group who exchanges PPS messages. This information is defined by platform
101 designer to provide all available domain documents, domain objects and domain properties.

102 **Domain document**

103 Document that is a content of message sent or received between application programs, and is
104 processed by a transaction. Domain document consists of a verb part and a noun part. Verbs
105 such as add, change and remove affect the types of messages, while nouns represented by
106 domain objects show the classes of domain objects. Specific classes of domain documents can
107 be defined by platform designer to share the domain information.

108 **Domain object**

109 Object necessary for representing production planning and scheduling information in
110 manufacturing operations management. Domain objects are contents of a domain document, and
111 represented by primitive elements. Specific classes of domain objects can be defined by platform
112 designer to share the domain information.

113 **Domain property**

114 Any parameters that show a property of a domain object. A domain property is represented by
115 XML attributes of the primitive element, or XML child elements of the primitive elements. A
116 domain object may have multiple domain properties that has same property name. Specific
117 properties of domain objects can be defined by platform designer to share the domain
118 information, and additionally defined by each application designer.

119 **Implementation profile**

120 Specification of capability of an application program in terms of exchanging PPS messages. The
121 profile includes a list of available documents and their properties that may be exchanged in PPS
122 messages among production planning and scheduling applications.

123 **Messaging model**

124 Simple patterns of messaging between sender and receiver, or requester and responder. Four
125 message models: NOTIFY, PUSH, PULL, SYNC are defined from an application independent
126 perspective.

127 **Primitive element**

128 XML element that represents a primitive object in the production planning and scheduling domain.
129 Nine primitive elements are defined in this specification. Every domain objects are represented by
130 the primitive elements.

131 **Transaction element**

132 XML element that represents a transaction to process message documents which is sent or
133 received between application programs. Transaction element can control a transaction process of
134 application program database by commitment and rollback. Transaction element may request
135 confirmation from receiver if the message has been received properly.

2 Core Elements

2.1 Primitive Elements

2.1.1 Structure of primitive elements

Primitive elements are the minimum series of element that corresponds to the most basic domain objects. The type of this element MUST be represented with the following XML schema.

```
<xsd:complexType name="PrimitiveType">
  <xsd:sequence>
    <xsd:element ref="Compose" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Produce" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Consume" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Assign" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Relation" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Location" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Capacity" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Progress" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Spec" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Start" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="End" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Event" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Price" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Cost" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="key" type="xsd:long"/>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="parent" type="xsd:string"/>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="status" type="xsd:string"/>
  <xsd:attribute name="party" type="xsd:string"/>
  <xsd:attribute name="plan" type="xsd:string"/>
  <xsd:attribute name="order" type="xsd:string"/>
  <xsd:attribute name="item" type="xsd:string"/>
  <xsd:attribute name="resource" type="xsd:string"/>
  <xsd:attribute name="process" type="xsd:string"/>
  <xsd:attribute name="lot" type="xsd:string"/>
  <xsd:attribute name="task" type="xsd:string"/>
  <xsd:attribute name="operation" type="xsd:string"/>
</xsd:complexType>
```

- *id* attribute SHOULD represent an identifier of the domain object.
- *key* attribute represents a key used in the local applications.
- *name* attribute represents the name of the domain object.
- *parent* attribute represents the identifier of the inherited object of the domain object.
- *type* attribute represents the modifier of the domain object.
- *status* attribute represents the status of the domain object.
- *party* attribute represents an identifier of the party associated with the domain object.
- *plan* attribute represents the identifier of the plan associated with the domain object.
- *order* attribute represents the identifier of the order associated with the domain object.

- 190 • *item* attribute represents the identifier of the item associated with the domain object.
- 191 • *resource* attribute represents the identifier of the resource associated with the domain object.
- 192 • *process* attribute represents the identifier of the process associated with the domain object.
- 193 • *lot* attribute represents the identifier of the lot associated with the domain object.
- 194 • *task* attribute represents the identifier of the task associated with the domain object.
- 195 • *operation* attribute represents the identifier of the operation associated with the domain object.
- 196
- 197 • *Compose* element represents the element corresponding to part of the domain object.
- 198 • *Produce* element represents the relation that the domain object produces.
- 199 • *Consume* element represents the relation that the domain object consumes.
- 200 • *Assign* element represents the relation that the domain object uses.
- 201 • *Relation* element represents the relation to other primitive elements.
- 202 • *Location* element represents the location where the domain object exists.
- 203 • *Capacity* element represents the capacity status of the domain object.
- 204 • *Progress* element represents the progress of the domain object.
- 205 • *Spec* element represents the specification of the domain object.
- 206 • *Start* element represents the start event of the domain object.
- 207 • *End* element represents the completion event of the domain object.
- 208 • *Event* element represents the optional event under the domain object.
- 209 • *Price* element represents the price of the domain object.
- 210 • *Cost* element represents the cost of the domain object.
- 211 • *Priority* element represents the priority of the domain object.
- 212 • *Display* element represents how to display the domain object.
- 213 • *Description* element represents the description of the domain object.
- 214 • *Author* element represents the author of the domain object information.
- 215 • *Date* element represents the date of the domain object information.

216 2.1.2 List of primitive elements

217 This specification defines nine primitive elements: *Party*, *Plan*, *Order*, *Item*, *Resource*, *Process*, *Lot*, *Task*,
 218 and *Operation*. The type of those elements MUST be represented with the following XML schema.

219

```

220 <xsd:element name="Party" type="PrimitiveType"/>
221 <xsd:element name="Plan" type="PrimitiveType"/>
222 <xsd:element name="Order" type="PrimitiveType"/>
223 <xsd:element name="Item" type="PrimitiveType"/>
224 <xsd:element name="Resource" type="PrimitiveType"/>
225 <xsd:element name="Process" type="PrimitiveType"/>
226 <xsd:element name="Lot" type="PrimitiveType"/>
227 <xsd:element name="Task" type="PrimitiveType"/>
228 <xsd:element name="Operation" type="PrimitiveType"/>
  
```

229

230 2.1.2.1 Party element

231 *Party* element represents a customer or a supplier. Customer is an object that requests some products or
 232 services to the enterprise. The requests are sent to a person who is in charge of production planning and

233 scheduling. Supplier is an object providing some products or services to the enterprise. Supplier receives
234 orders from the enterprise, and provides corresponding items, resources or processes for the enterprise.

235 2.1.2.2 Plan element

236 *Plan* element represents a value planned for particular products or services. The value shows volume of
237 the products or services required or resulted during certain period of time. Typical cases of planning
238 period include day, week and month.

239 2.1.2.3 Order element

240 *Order* element represents an object of information produced to request some products or services. Order
241 is source to create production orders that are finally dispatched to the plant floor. Orders can be divided
242 into inventory order, capacity order and production order according to the type of request.

243

244 Example: Item "A" products are requested.

```
245 <Order id="Z01" item="A">  
246   <Spec type="quantity"><Qty value="10"/></Spec>  
247 </Order>
```

248 Example: Three labors in "group B" are requested.

```
249 <Order id="Z02" resource="groupB">  
250   <Spec type="quantity"><Qty value="3"/></Spec>  
251 </Order>
```

252 Example: Switching operation is requested two times.

```
253 <Order id="Z03" process="change01">  
254   <Spec type="quantity"><Qty value="2"/></Spec>  
255 </Order>
```

256 Example: Order which consist of 10 of "A" and 5 of "B" is totally 3,000 yen.

```
257 <Order id="Z00">  
258   <Compose order="Z01"/>  
259   <Compose order="Z02"/>  
260   <Price value="3000" unit="yen"/>  
261 </Order>  
262 <Order id="Z01" item="A">  
263   <Spec type="quantity"><Qty value="10"/></Spec>  
264 </Order>  
265 <Order id="Z02" item="B">  
266   <Spec type="quantity"><Qty value="5"/></Spec>  
267 </Order>
```

268

269 2.1.2.4 Item element

270 *Item* element represents a product, component, parts, work in process (WIP), raw material and other
271 items. Item is produced by any processes, and after that, it is consumed by another processes.

272 2.1.2.5 Resource element

273 *Resource* element represents a resource, which is an object enabling production, transportation, storage,
274 inspection and other various services. As resource can produce tasks to execute operations, it is
275 assigned to an operation by considering its volume of capacity.

276 2.1.2.6 Process element

277 *Process* element represents a process that has a function to produce value. Process can be defined as a
278 segment of activities in production process. It produces and consumes production items by being
279 executed during certain period of time.

280 2.1.2.7 Lot element

281 *Lot* element represents a production lot. Production lot is an object corresponding to a concrete item that
282 actually exists in a specific place at a specific date and time. Lot is produced by an operation and finally
283 consumed by another operation or discarded.

284 2.1.2.8 Task element

285 *Task* element represents a task, which is an object showing the usage of a specific resource capability for
286 a specific period of time. Schedule may request a certain volume of task for each resource assigned to
287 execute the appropriate operations.

288

289 Example: Task corresponding to the volume that 3 labors work load is required for 2 days

```
290 <Task id="T01">  
291   <Capacity type="human"><Qty value="3"/></Capacity>  
292   <Capacity type="duration"><Qty value="2" unit="day" /></Capacity>  
293 </Task>
```

294

295 2.1.2.9 Operation element

296 *Operation* element represents a segment of activities that is actually dispatched to plant floor. Operation
297 identifies an executable function at a specific place on a plant floor for a specific time. Operation is
298 associated with a specific lot and task by executing those activities.

299 2.2 Relational Elements

300 2.2.1 Structure of relational elements

301 *Relational* elements represent any relations between domain objects. A relational element can have
302 properties. The type of this element MUST be represented with the following XML schema.

303

```
304 <xsd:complexType name="RelationalType">  
305   <xsd:sequence>  
306     <xsd:element ref="Location" minOccurs="0" maxOccurs="unbounded"/>  
307     <xsd:element ref="Capacity" minOccurs="0" maxOccurs="unbounded"/>  
308     <xsd:element ref="Progress" minOccurs="0" maxOccurs="unbounded"/>  
309     <xsd:element ref="Spec" minOccurs="0" maxOccurs="unbounded"/>  
310     <xsd:element ref="Start" minOccurs="0" maxOccurs="unbounded"/>  
311     <xsd:element ref="End" minOccurs="0" maxOccurs="unbounded"/>  
312     <xsd:element ref="Event" minOccurs="0" maxOccurs="unbounded"/>  
313     <xsd:element ref="Price" minOccurs="0" maxOccurs="unbounded"/>  
314     <xsd:element ref="Cost" minOccurs="0" maxOccurs="unbounded"/>  
315     <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>  
316     <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>  
317     <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>  
318     <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>  
319     <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>  
320     <xsd:element ref="Qty" minOccurs="0" maxOccurs="unbounded"/>  
321     <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>  
322     <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>  
323   </xsd:sequence>  
324   <xsd:attribute name="id" type="xsd:string"/>  
325   <xsd:attribute name="key" type="xsd:long"/>  
326   <xsd:attribute name="name" type="xsd:string"/>  
327   <xsd:attribute name="type" type="xsd:string"/>  
328   <xsd:attribute name="status" type="xsd:string"/>  
329   <xsd:attribute name="apply" type="xsd:string"/>  
330   <xsd:attribute name="party" type="xsd:string"/>  
331   <xsd:attribute name="plan" type="xsd:string"/>  
332   <xsd:attribute name="order" type="xsd:string"/>  
333   <xsd:attribute name="item" type="xsd:string"/>
```

```
334 <xsd:attribute name="resource" type="xsd:string"/>
335 <xsd:attribute name="process" type="xsd:string"/>
336 <xsd:attribute name="lot" type="xsd:string"/>
337 <xsd:attribute name="task" type="xsd:string"/>
338 <xsd:attribute name="operation" type="xsd:string"/>
339 </xsd:complexType>
```

- 340
- 341 • *id* attribute SHOULD represent an identifier of the relation.
 - 342 • *key* attribute represents a key used in the local applications.
 - 343 • *name* attribute represents the name of the relation.
 - 344 • *type* attribute represents the modifier of the relation.
 - 345 • *status* attribute represents the status of the relation.
 - 346 • *apply* attribute represents application type of the relation. This element is a disjunctive (OR) content
 - 347 under the parent object, if the attribute value is "*disjunctive*".
 - 348 • *party* attribute represents an identifier of the party associated with the relation.
 - 349 • *plan* attribute represents the identifier of the plan associated with the relation.
 - 350 • *order* attribute represents the identifier of the order associated with the relation.
 - 351 • *item* attribute represents the identifier of the item associated with the relation.
 - 352 • *resource* attribute represents the identifier of the resource associated with the relation.
 - 353 • *process* attribute represents the identifier of the process associated with the relation.
 - 354 • *lot* attribute represents the identifier of the lot associated with the relation.
 - 355 • *task* attribute represents the identifier of the task associated with the relation.
 - 356 • *operation* attribute represents the identifier of the operation associated with the relation.
- 357
- 358 • *Location* element represents the location associated with the relation.
 - 359 • *Capacity* element represents the capacity status of the relation.
 - 360 • *Progress* element represents the progress of the relation.
 - 361 • *Spec* element represents the specification of the relation.
 - 362 • *Start* element represents the start event of the relation.
 - 363 • *End* element represents the completion event of the relation.
 - 364 • *Event* element represents the optional event under the relation.
 - 365 • *Price* element represents the price of the relation.
 - 366 • *Cost* element represents the cost of the relation.
 - 367 • *Priority* element represents the priority of the relation.
 - 368 • *Display* element represents how to display the relation.
 - 369 • *Description* element represents the description of the relation.
 - 370 • *Author* element represents the author of the relation information.
 - 371 • *Date* element represents the date of the relation information.
 - 372 • *Qty* element represents the quantity of the relation.
 - 373 • *Char* element represents the qualitative value of the relation.
 - 374 • *Time* element represents the time of the relation.
- 375

376 2.2.2 List of relational elements

377 This part of specifications defines five relational elements: *Compose*, *Produce*, *Consume*, *Assign*, and
378 *Relation*. Relational element defines relationship between the parent element and those that characterize
379 the element. The type of this element MUST be represented with the following XML schema.

380

```
381 <xsd:element name="Compose" type="RelationalType"/>  
382 <xsd:element name="Produce" type="RelationalType"/>  
383 <xsd:element name="Consume" type="RelationalType"/>  
384 <xsd:element name="Assign" type="RelationalType"/>  
385 <xsd:element name="Relation" type="RelationalType"/>
```

386

387 2.2.2.1 Compose element

388 *Compose* element defines a hierarchical relation between the parent element and another same primitive
389 element that addresses one level upper or lower than the target element. This element can represent that
390 the object referred to in this element composes or be composed by the parent element.

391

392 Example: Product "A" family includes product "A1" and product "A2".

```
393 <Item id="A">  
394   <Compose type="child" item="A1"/>  
395   <Compose type="child" item="A2"/>  
396 </Item>
```

397 Example: Product "B" is assembled with 2 of parts "C1" and 3 of parts "C2".

```
398 <Item id="B">  
399   <Compose type="child" item="C1"><Qty value="2"/></Compose>  
400   <Compose type="child" item="C2"><Qty value="3"/></Compose>  
401 </Item>
```

402 Example: 2 of parts "C1" are used for product "B1", and 5 of parts "C1" are used for product "B2".

```
403 <Item id="C1">  
404   <Compose type="parent" item="B1"><Qty value="2"/></Compose>  
405   <Compose type="parent" item="B2"><Qty value="5"/></Compose>  
406 </Item>
```

407

408 2.2.2.2 Produce element

409 *Produce* element defines a relation between processes and items, or a relation between operations and
410 lots. This element can show the quantity of the item or lot produced by the process or operation
411 respectively, or how many items or lots are produced by the process or the operation respectively.

412 2.2.2.3 Consume element

413 *Consume* element defines a relation between processes and items, or a relation between operations and
414 lots. This element can show the quantity of the item or lot consumed by the process or operation
415 respectively, or how many items or lots are consumed by the process or operation respectively.

416 2.2.2.4 Assign element

417 *Assign* element defines a relation between processes and resources, or a relation between operations
418 and tasks. This element can show the volume of capacity provided by the resource or task assigned for
419 the process or operation respectively, or how many resources or tasks are used.

420 2.2.2.5 Relation element

421 *Relation* element can show that the parent element has a specific relation to other primitive elements.
422 This element can additionally define relational classes between primitive elements. Examples include
423 precedence relations and pegging relations.

424 2.3 Specific Elements

425 2.3.1 Structure of specific element

426 Specific elements are defined to represent any properties of the primitive element. This element MAY be
427 described more than once on the same parent element if the value is historical. Those multiple properties
428 have time stamp. The type of this element MUST be represented with the following XML schema.

429

```
430 <xsd:complexType name="SpecificType">
431   <xsd:sequence>
432     <xsd:element ref="Start" minOccurs="0" maxOccurs="unbounded"/>
433     <xsd:element ref="End" minOccurs="0" maxOccurs="unbounded"/>
434     <xsd:element ref="Event" minOccurs="0" maxOccurs="unbounded"/>
435     <xsd:element ref="Price" minOccurs="0" maxOccurs="unbounded"/>
436     <xsd:element ref="Cost" minOccurs="0" maxOccurs="unbounded"/>
437     <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
438     <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
439     <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
440     <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
441     <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
442     <xsd:element ref="Qty" minOccurs="0" maxOccurs="unbounded"/>
443     <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>
444     <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>
445   </xsd:sequence>
446   <xsd:attribute name="id" type="xsd:string"/>
447   <xsd:attribute name="key" type="xsd:long"/>
448   <xsd:attribute name="name" type="xsd:string"/>
449   <xsd:attribute name="type" type="xsd:string"/>
450   <xsd:attribute name="status" type="xsd:string"/>
451   <xsd:attribute name="apply" type="xsd:string"/>
452 </xsd:complexType>
```

453

- 454 • *id* attribute SHOULD represent an identifier of the property.
- 455 • *key* attribute represents a key used in the local applications.
- 456 • *name* attribute represents the name of the property.
- 457 • *type* attribute represents the modifier of the property.
- 458 • *status* attribute represents the status of the property.
- 459 • *apply* attribute represents application type of the property. The value of the element is relative, if the
460 value is “*relative*”.
- 461
- 462 • *Start* element represents the start event of the property.
- 463 • *End* element represents the completion event of the property.
- 464 • *Event* element represents the optional event under the property.
- 465 • *Price* element represents the price of the property.
- 466 • *Cost* element represents the cost of the property.
- 467 • *Priority* element represents the priority of the property.
- 468 • *Display* element represents how to display the property.
- 469 • *Description* element represents the description of the property.

- 470 • *Author* element represents the author of the property information.
- 471 • *Date* element represents the date of the property information.
- 472 • *Qty* element represents the quantity of the property.
- 473 • *Char* element represents the qualitative value of the property.
- 474 • *Time* element represents the time of the property.

475

476 2.3.2 List of specific elements

477 For specific elements, this part of specifications has four elements: *Location*, *Capacity*, *Progress*, and
 478 *Spec*. The type of this element MUST be represented with the following XML schema.

479

```
480 <xsd:element name="Location" type="SpecificType"/>
481 <xsd:element name="Capacity" type="SpecificType"/>
482 <xsd:element name="Progress" type="SpecificType"/>
483 <xsd:element name="Spec" type="SpecificType"/>
```

484

485 2.3.2.1 Location element

486 *Location* element represents a location. When the expression of location has structure, multiple values
 487 can be set by describing different names of the data. Change of the location depending on time can also
 488 be represented by multiple values.

489

490 Example: Customer's address

```
491 <Party id="ABC Inc.">
492   <Location type="address"><Char value="123 ABC street"/></Location>
493   <Location type="city"><Char value="Cambridge"/></Location>
494   <Location type="state"><Char value="MA"/></Location>
495   <Location type="code"><Char value="02139"/></Location>
496   <Location type="country"><Char value="USA"/></Location>
497 </Party>
```

498

499 2.3.2.2 Capacity element

500 *Capacity* element represents volume of capability provided by resources, items or processes. In the case
 501 of resource capability, it may show available amount of corresponding tasks. In the case of Items, it
 502 shows the available amount of Lots. And for Processes, it shows maximum ratio of production. All of this
 503 information is represented in a time horizon.

504

505 Example: Inventory level of "material01"

```
506 <Item id="material01">
507   <Capacity><Qty value="150"/></Capacity>
508 </Item>
```

509 Example: Temporal change of the material

```
510 <Item id="material01">
511   <Capacity><Qty value="150"><Time value="2005-04-10T00:00:00"/></Capacity>
512   <Capacity><Qty value="200"><Time value="2005-04-17T00:00:00"/></Capacity>
513 </Item>
```

514 Example: Material location information: Stock of "material01" is 150 located at "storage01"

```
515 <Item id="material01">
```



```
516 <Location value="storage01"/>
517 <Capacity><Qty value="150"/></Capacity>
518 </Item>
```

519

520 2.3.2.3 Progress element

521 *Progress* element represents progress of order and operation, or status of lot and task. This element
522 shows the latest data, status or progress at a specific time point. This element MAY represent a change
523 of time-dependent values.

524 2.3.2.4 Spec element

525 *Spec* element represents various specifications for primitive elements. The content can be represented
526 with a pair of a spec name and a value. This element can also represent time-dependent change of the
527 value. The value of the specification is represented with one data type of a numerical value, characters
528 and date time.

529 2.4 Eventual Elements

530 2.4.1 Structure of eventual element

531 Eventual elements represent any properties that occur at one time point. Any type of events can be
532 specified by using this element. The type of this element MUST be represented with the following XML
533 schema.

534

```
535 <xsd:complexType name="EventualType">
536   <xsd:sequence>
537     <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>
538     <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>
539     <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
540     <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
541     <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
542     <xsd:element ref="Qty" minOccurs="0" maxOccurs="unbounded"/>
543     <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>
544     <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>
545   </xsd:sequence>
546   <xsd:attribute name="id" type="xsd:string"/>
547   <xsd:attribute name="key" type="xsd:long"/>
548   <xsd:attribute name="name" type="xsd:string"/>
549   <xsd:attribute name="type" type="xsd:string"/>
550   <xsd:attribute name="status" type="xsd:string"/>
551   <xsd:attribute name="apply" type="xsd:string"/>
552   <xsd:attribute name="condition" type="xsd:string"/>
553   <xsd:attribute name="value" type="xsd:string"/>
554 </xsd:complexType>
```

555

- 556 • *id* attribute SHOULD represent an identifier of the property.
- 557 • *key* attribute represents a key used in the local applications.
- 558 • *name* attribute represents the name of the property.
- 559 • *type* attribute represents the modifier of the property.
- 560 • *status* attribute represents the status of the property.
- 561 • *apply* attribute represents application type of the property. The value of this element is exclusive, if
562 the value is “exclusive”.
- 563 • *condition* attribute represents the condition of the property.
- 564 • *value* attribute represents the qualitative value of the property.

565
566
567
568
569
570
571
572
573
574

- *Priority* element represents the priority of the property.
- *Display* element represents how to display the property.
- *Description* element represents the description of the property.
- *Author* element represents the author of the property information.
- *Date* element represents the date of the property information.
- *Qty* element represents the quantity of the property.
- *Char* element represents the qualitative value of the property.
- *Time* element represents the time of the property.

575 2.4.2 List of eventual elements

576 This part of specifications defines three eventual elements: *Start*, *End*, and *Event*. The *Start* and *End* are
577 special cases of *Event* element. The type of this element MUST be represented with the following XML
578 schema.

579

```
580 <xsd:element name="Start" type="EventualType"/>  
581 <xsd:element name="End" type="EventualType"/>  
582 <xsd:element name="Event" type="EventualType"/>
```

583

584 2.4.2.1 Start element

585 *Start* element represents a start event of orders, processes or operations. In case of order, this element
586 represents an event at the earliest start time of corresponding operations.

587 2.4.2.2 End element

588 *End* element represents an end event of orders, processes or operations. In case of order, this element
589 represents an event at the latest end time of corresponding operations.

590 2.4.2.3 Event element

591 *Event* element represents an event associated with a customer, supplier, item, resource, process or
592 operation. Event brings any action or any status change at a specific time point. In general, the status
593 value of item or resource changes discontinuously before the event.

594

595 2.5 Accounting Elements

596 2.5.1 Structure of Accounting element

597 Accounting element represents any accounting information such as profit revenue and cost spending.
598 Price and cost associated with goods and services are the target of the elements. The type of this
599 element MUST be represented with the following XML schema.

600

```
601 <xsd:complexType name="AccountingType">  
602   <xsd:sequence>  
603     <xsd:element ref="Priority" minOccurs="0" maxOccurs="unbounded"/>  
604     <xsd:element ref="Display" minOccurs="0" maxOccurs="unbounded"/>  
605     <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>  
606     <xsd:element ref="Author" minOccurs="0" maxOccurs="unbounded"/>
```

```

607     <xsd:element ref="Date" minOccurs="0" maxOccurs="unbounded"/>
608     <xsd:element ref="Qty" minOccurs="0" maxOccurs="unbounded"/>
609     <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>
610     <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>
611   </xsd:sequence>
612   <xsd:attribute name="id" type="xsd:string"/>
613   <xsd:attribute name="key" type="xsd:long"/>
614   <xsd:attribute name="name" type="xsd:string"/>
615   <xsd:attribute name="type" type="xsd:string"/>
616   <xsd:attribute name="status" type="xsd:string"/>
617   <xsd:attribute name="value" type="xsd:string"/>
618   <xsd:attribute name="condition" type="xsd:string"/>
619   <xsd:attribute name="apply" type="xsd:string"/>
620 </xsd:complexType>

```

- 621
- 622 • *id* attribute SHOULD represent an identifier of the property.
- 623 • *key* attribute represents a key used in the local applications.
- 624 • *name* attribute represents the name of the property.
- 625 • *type* attribute represents the modifier of the property.
- 626 • *status* attribute represents the status of the property.
- 627 • *apply* attribute represents application type of the property. The value of this element is exclusive, if
- 628 the value is “exclusive”.
- 629 • *condition* attribute represents the condition of the property.
- 630 • *value* attribute represents the qualitative value of the property.
- 631
- 632 • *Priority* element represents the priority of the property.
- 633 • *Display* element represents how to display the property.
- 634 • *Description* element represents the description of the property.
- 635 • *Author* element represents the author of the property information.
- 636 • *Date* element represents the date of the property information.
- 637 • *Qty* element represents the quantitative value of the property.
- 638 • *Char* element represents the qualitative value of the property.
- 639 • *Time* element represents the temporal value of the property.
- 640

641 2.5.2 List of accounting elements

642 For accounting elements, *Price* element and *Cost* element are defined in this specification. The type of
643 this element MUST be represented with the following XML schema.

```

644
645     <xsd:element name="Price" type="AccountingType"/>
646     <xsd:element name="Cost" type="AccountingType"/>

```

647

648 2.5.2.1 Price element

649 *Price* element represents a price. This element can be used to represent price information of primitive
650 element and some properties.

651 2.5.2.2 Cost element

652 *Cost* element represents a cost. This element can be used to represent cost information of primitive
653 element and some properties.

654

655 2.6 Administrative Elements

656 2.6.1 Structure of Administrative Elements

657 Administrative elements represent any administrative information, which is not the main body of the
658 problem domain but the information how to deal with the domain information. The type of this element
659 MUST be represented with the following XML schema.

660

```
661 <xsd:complexType name="AdministrativeType">  
662   <xsd:sequence>  
663     <xsd:element ref="Qty" minOccurs="0" maxOccurs="unbounded"/>  
664     <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>  
665     <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>  
666   </xsd:sequence>  
667   <xsd:attribute name="name" type="xsd:string"/>  
668   <xsd:attribute name="type" type="xsd:string"/>  
669   <xsd:attribute name="status" type="xsd:string"/>  
670   <xsd:attribute name="apply" type="xsd:string"/>  
671   <xsd:attribute name="condition" type="xsd:string"/>  
672   <xsd:attribute name="value" type="xsd:string"/>  
673 </xsd:complexType>
```

674

- 675 • *name* attribute represents the name of the property.
 - 676 • *type* attribute represents the modifier of the property.
 - 677 • *status* attribute represents the status of the property.
 - 678 • *apply* attribute represents application type of the property. The value of this element is exclusive, if
679 the value is “exclusive”.
 - 680 • *condition* attribute represents the condition of the property.
 - 681 • *value* attribute represents the qualitative value of the property.
-
- 683 • *Qty* element represents the quantitative value of the property.
 - 684 • *Char* element represents the qualitative value of the property.
 - 685 • *Time* element represents the temporal value of the property.

686

687 2.6.2 List of Administrative Elements

688 For administrative elements, *Priority*, *Display*, *Description*, *Author* and *Date* elements are defined in this
689 specification. The type of this element MUST be represented with the following XML schema.

690

```
691 <xsd:element name="Priority" type="AdministrativeType"/>  
692 <xsd:element name="Display" type="AdministrativeType"/>  
693 <xsd:element name="Description" type="AdministrativeType"/>  
694 <xsd:element name="Author" type="AdministrativeType"/>  
695 <xsd:element name="Date" type="AdministrativeType"/>
```

696

697 **2.6.2.1 Priority element**

698 *Priority* element represents the priority of the primitive element or the parent element. This information is
699 used to make a decision for planning or scheduling.

700 **2.6.2.2 Display element**

701 *Display* element is an element to set how to display the parent element. This element can specify colors
702 or display locations on the screen.

703 **2.6.2.3 Description element**

704 *Description* element is an element to set an optional comment of the parent element. The comment data
705 type is a character string.

706 **2.6.2.4 Author element**

707 *Author* element represents the author and its related information such as the authoring date. This
708 information is not about the target domain model, but information processing model.

709 **2.6.2.5 Date element**

710 *Date* element is an element that shows the creation date, expire date, revising date, and so forth. This
711 information is for administrative use of the domain model.

712 **2.7 Data Elements**

713 **2.7.1 Qty element**

714 *Qty* element SHOULD represent quantitative information. This element can be used to represent the
715 quantitative numerical data by decimal type data format. Unit of the value can be set in this element, and
716 representation of fraction is available. The type of this element MUST be represented with the following
717 XML schema.

718

```
719 <xsd:element name="Qty">  
720   <xsd:complexType>  
721     <xsd:attribute name="name" type="xsd:string"/>  
722     <xsd:attribute name="type" type="xsd:string"/>  
723     <xsd:attribute name="status" type="xsd:string"/>  
724     <xsd:attribute name="apply" type="xsd:string"/>  
725     <xsd:attribute name="condition" type="xsd:string"/>  
726     <xsd:attribute name="value" type="xsd:decimal"/>  
727     <xsd:attribute name="count" type="xsd:long"/>  
728     <xsd:attribute name="unit" type="xsd:string"/>  
729     <xsd:attribute name="base" type="xsd:decimal"/>  
730   </xsd:complexType>  
731 </xsd:element>
```

732

- 733 • *name* attribute represents the name of the data.
- 734 • *type* attribute represents the modifier of the data.
- 735 • *status* attribute represents the status of the data.
- 736 • *apply* attribute represents application type of the data. The value of this element is exclusive, if the
737 value is “exclusive”.
- 738 • *condition* attribute represents the condition of the data.
- 739 • *value* attribute represents the content corresponding to the qty element.
- 740 • *count* attribute represents the countable value of the data.

- 741
- *unit* attribute represents the type of unit of the data.
- 742
- *base* attribute represents the base data of the data. The value of the “value” attribute is divided with
- 743 this value.

744

745 Example: 1/3 meters

```
746 <Qty value="1" unit="m" base="3"/>
```

747 Example: 3 weeks (discrete time scale)

```
748 <Qty count="3" unit="week" />
```

749

750 2.7.2 Char element

751 *Char* element SHOULD represent character data. This element can be used to represent a qualitative
752 value of specification or a value of location. The type of this element MUST be represented with the
753 following XML schema.

754

```
755 <xsd:element name="Char">  
756   <xsd:complexType>  
757     <xsd:attribute name="name" type="xsd:string"/>  
758     <xsd:attribute name="type" type="xsd:string"/>  
759     <xsd:attribute name="status" type="xsd:string"/>  
760     <xsd:attribute name="apply" type="xsd:string"/>  
761     <xsd:attribute name="condition" type="xsd:string"/>  
762     <xsd:attribute name="value" type="xsd:string"/>  
763     <xsd:attribute name="count" type="xsd:long"/>  
764     <xsd:attribute name="unit" type="xsd:string"/>  
765     <xsd:attribute name="base" type="xsd:string"/>  
766   </xsd:complexType>  
767 </xsd:element>
```

768

- 769 • *name* attribute represents the name of the data.
- 770 • *type* attribute represents the modifier of the data.
- 771 • *status* attribute represents the status of the data.
- 772 • *apply* attribute represents application type of the data. The value of this element is exclusive, if the
773 value is “exclusive”.
- 774 • *condition* attribute represents the condition of the data.
- 775 • *value* attribute represents the content corresponding to the data.
- 776 • *count* attribute represents the countable value of the data.
- 777 • *unit* attribute represents the type of unit of the data.
- 778 • *base* attribute represents the base data of the data. The value of the “value” attribute is divided with
779 this value.

780

781 2.7.3 Time element

782 *Time* element SHOULD represent a specific time. Time is represented by a continuous time scale, or a
783 specific discrete time scale. The type of this element MUST be represented with the following XML
784 schema.

785

```
786 <xsd:element name="Time">
```

```
787 <xsd:complexType>
788 <xsd:attribute name="name" type="xsd:string"/>
789 <xsd:attribute name="type" type="xsd:string"/>
790 <xsd:attribute name="status" type="xsd:string"/>
791 <xsd:attribute name="apply" type="xsd:string"/>
792 <xsd:attribute name="condition" type="xsd:string"/>
793 <xsd:attribute name="value" type="xsd:dateTime"/>
794 <xsd:attribute name="count" type="xsd:long"/>
795 <xsd:attribute name="unit" type="xsd:string"/>
796 <xsd:attribute name="base" type="xsd:dateTime"/>
797 </xsd:complexType>
798 </xsd:element>
```

799

- 800 • *name* attribute represents the name of the data.
- 801 • *type* attribute represents the modifier of the data.
- 802 • *status* attribute represents the status of the data.
- 803 • *apply* attribute represents application type of the data. The value of this element is exclusive, if the
804 value is “exclusive”.
- 805 • *condition* attribute represents the condition of the data.
- 806 • *value* attribute represents the content corresponding to the data.
- 807 • *count* attribute represents the countable value of the data.
- 808 • *unit* attribute represents the type of unit of the data.
- 809 • *base* attribute represents the base data of the data. The value of the “value” attribute is divided with
810 this value.

811

812 Example: noon on May 13th, 2005

```
813 <Time value="2005-05-13T12:00:00"/>
```

814 Example: 2 months later since the present month (May, 2005) (discrete time scale)

```
815 <Time count="2" unit="month" base="2005-05-01T00:00:00"/>
```

816

3 Transaction Messages

817

3.1 Messaging model

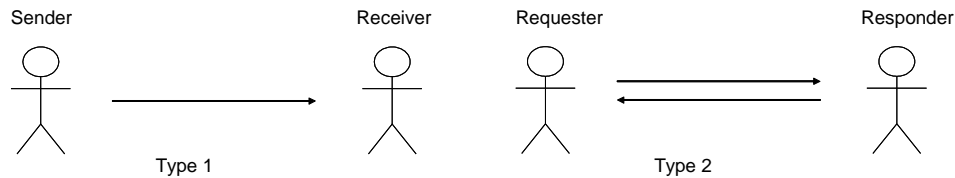
818

3.1.1 Basic Unit of messaging

819

Two basic unit of messaging are defined in this specification. The first one is a communication between sender and receiver (Type 1), where the sender simply sends a message to the receiver without any negotiations. The second is a communication between requester and responder (Type 2), where the requester asks the responder to do some services. The responder may answer to the sender by sending appropriate message. The responding message is mandatory or optional depending on the service. The receiver or responder may be multiple at one transaction, so as to make broad casting.

825



826

827 *Figure 3.1 Basic unit of messaging*

828

829

The basic units used to define several messaging models in later sections. However in many practical business situations, communication protocols such as customer negotiation with price and due dates, communication procedures are designed using these basic patterns as a building block. In such cases, how to combine the component is not in the scope of this standard.

833

In addition, underlying communication protocols such as HTTP and TCP/IP may used to define for the simple messaging unit, considering security, reliability, efficiency and so forth. In such cases, messages may be sent several times for the one arrow in Figure 3.1. This is also not in the scope of this part.

836

Application programs communicate using the basic unit of messaging to perform particular business logics. One or more than one transactions of domain documents are contained in each message.

837

838

3.1.2 Message classes

839

Domain documents, which are exchanged between sender and receiver, or requester and responder, are defined for each transaction. According to the verb information of each document, they can be categorized into 8 different classes. The table shows the message types.

842

843

Table 3.1 Action classes of domain documents

Action classes	Description
Add	The message requests to add the specified domain objects to the database managed by the responder.
Change	The message requests to change the specified domain objects in the database managed by the responder.
Remove	The message requests to remove the specified domain objects in the database managed by the responder.
Confirm	The message responds from the responder to the requester as a confirmation of the

	results.
Notify	The message informs any domain objects to the receiver as a notification from the sender.
Sync	The message requests the owner of information to send notify message synchronously at the time the specified event occurs.
Get	The message asks the responder to show the specified domain objects in a specified format by responding Show message.
Show	The message responses the requested information of domain objects to the Get message from the requester.

844

845 In order to ask the confirmation from responders, domain documents that perform with Add, Change,
846 Remove or Sync action MAY have an attribute of the following confirmation requests.

847

848 *Table 3.2 Confirmation request*

Confirm type	Description
Never	Responder SHOULD NOT respond to the request.
OnError	Responder SHOULD respond to the request, only if any errors in processing the request occur.
Always	Responder SHOULD always respond to the request.

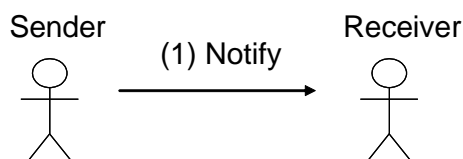
849

850 **3.1.3 Messaging models**

851 **3.1.3.1 NOTIFY model (Type 1)**

852 Basic messaging unit of Type 1 performs in the NOTIFY model. In this model, the sender sends a Notify
853 message to the receiver. There is no obligation on the receiver to respond to the message, nor to make a
854 task for the message.

855



856

857 *Figure3. 2 NOTIFY model*

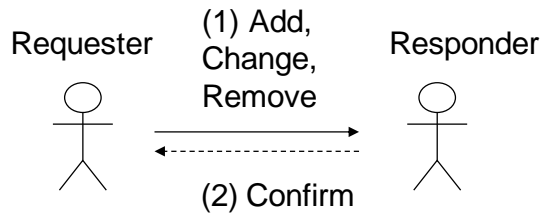
858

859 **3.1.3.2 PUSH model (Type 2)**

860 In PUSH model, domain document with Add action, Change action and Remove action can be requested
861 and processed by applications. This model is enabled by type 1 messaging unit.

862 In Add transaction, the requester sends an Add message to request responder to add the specified
863 domain objects to the database that is managed by the responder. After making the task of adding the
864 information, the responder can send a Confirm message depending on the confirmation request.

865



866
867 *Figure 3.3 PULL model*

868
869 Change transaction performs when the requester tries to change the specified domain objects in the
870 database that is managed by the responder. The requester sends a Change message to the responder
871 as a request to change. The responder can do the task and send a Confirm message as a result of the
872 task.

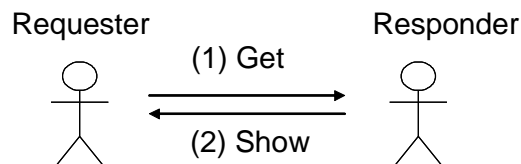
873 Remove transaction performs when the requester tries to delete the specified domain objects in the
874 database managed by the responder. The requester sends a Remove message, and the responder
875 responds a Confirm message if the Remove message has a confirmation request.

876 Responder processes the requested actions, and if necessary, responds confirmation documents to the
877 requester.

878 **3.1.3.3 PULL model (Type 2)**

879 PULL model is defined for one or more than one actions of Get-Show transactions. Get-Show transaction
880 performs like a query-response process in the client-server database systems. The requester sends a Get
881 message to the responder in order to get information of the specified domain objects. The responder tries
882 to answer the request by sending Show message with corresponding information which is managed by
883 the responder.

884



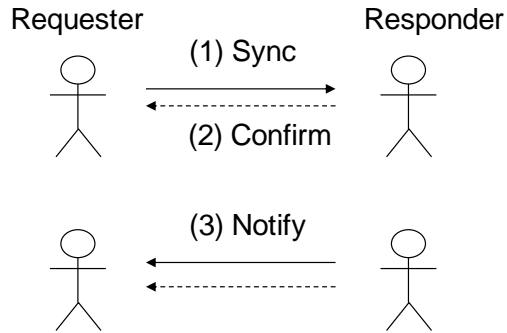
885
886 *Figure 3.4 PULL model*

887

888 **3.1.3.4 SYNC model (Type 2 and Type 1)**

889 SYNC model consists of a Sync transaction (Type 2) and several Notify transactions (Type 1). Sync
890 transaction performs that requester requests responder to send Notify message synchronously at the
891 time when the specified event occurs on the domain objects owned by the responder. Responder keeps
892 monitoring the event in order to send Notify messages by invoking another Notify transaction. Notify
893 messages are sent repetitively when the event occurs until the Sync request is canceled.

894



895

896 *Figure 3.5 SYNC model*

897 **3.1.4 Procedures on responders**

898 **3.1.4.1 Common tasks**

899 Responders SHOULD have capability to perform the following tasks when a message document is
900 received.

- 901 • The responder, who receives a proper Get document, SHOULD send a Show message to the
902 requester. The Show message SHOULD have either error information or domain object requested by
903 the requester in the specified forms.
- 904 • The responder, who receives a proper Add document, SHOULD add the domain objects in the
905 message to the database that is managed by the responder, unless the ID of the object already
906 exists.
- 907 • The responder, who receives a proper Change document, SHOULD change the target domain object
908 in the database managed by the responder to the new data in the message, unless the ID of the
909 object doesn't exist.
- 910 • The responder, who receives a proper Remove document, SHOULD delete the target domain object
911 in the database managed by the responder, unless the ID of the object doesn't exist.

912 **3.1.4.2 Confirm message**

913 The responder of Add, Change, Remove and Sync document SHOULD have capability to make the
914 following tasks when the message received has a confirmation request.

- 915 • The responder SHOULD send a Confirm document to the requester when the Add document
916 received has an attribute of confirm="Always". The Confirm document SHOULD have either error
917 information or the id list that shows all the objects added to the database.
- 918 • The responder SHOULD send a Confirm document to the requester when the Change document
919 received has an attribute of confirm="Always". The Confirm document SHOULD have either error
920 information or the id list that shows all the objects changed in the database.
- 921 • The responder SHOULD send a Confirm document to the requester when the Remove document
922 received has an attribute of confirm="Always". The Confirm document SHOULD have either error
923 information or the id list that shows all the objects deleted in the database.
- 924 • The responder SHOULD send a Confirm document to the requester when the Sync document
925 received has an attribute of confirm="Always". The Confirm document SHOULD have either error
926 information or the id list that shows all the objects to be monitored for synchronization.
- 927 • The responder SHOULD NOT send a Confirm document to the requester when the document
928 received has an attribute of confirm="Never".

929 **3.1.4.3 Error handling**

930 To deal with errors occurred during the process of document in responder application, e.g. syntax or
931 semantic problems detected by the responder's programs, the responder SHOULD have capability of the
932 following error handling:

- 933 • In PULL model, responder, who receives a Get document and is hard to respond in normal
934 processes because of errors, SHOULD send a Show document with the error information to the
935 requester.
- 936 • In PUSH model and SYNC model, responder who receives a document that has attribute of
937 confirm="OnError" or "Always" and detects errors during the process requested SHOULD send a
938 Confirm document with the error information to the requester.
- 939 • The responder SHOULD NOT send a Confirm document nor Show document to the requester when
940 the document received has an attribute of confirm="Never", even if there is an error.

941

942 **3.2 Add, Change and Remove (PUSH model)**

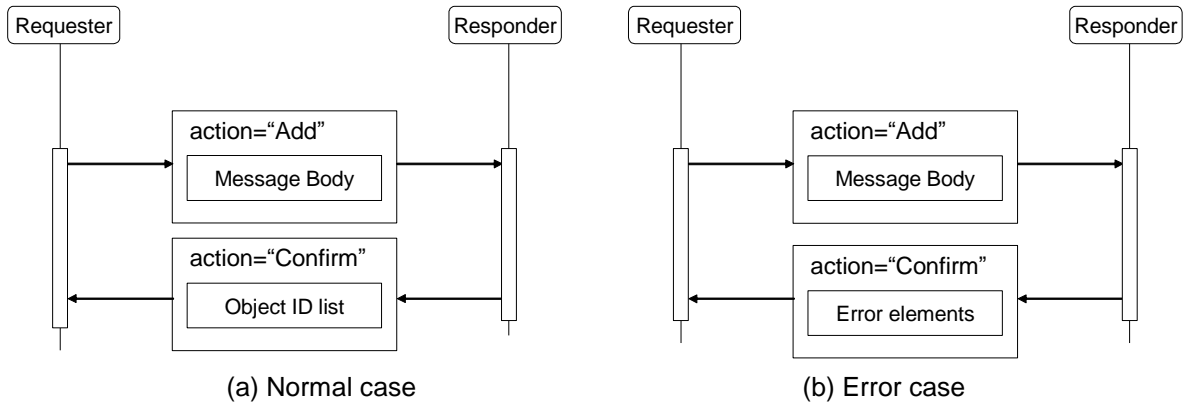
943 **3.2.1 Add transaction**

944 Add document requests the responder to add the specified domain objects in the document to the
945 database managed by the responder.

946 When the Add document request to add domain objects with ID specified at the "id" attribute, responder
947 SHOULD check existence of the ID in its database and add the data if the corresponding data does not
948 already exist in the database. If the document has an ID that already exists in the database, the
949 responder SHOULD NOT add the data.

950 When the Add document request to add domain object without ID, the responder SHOULD create any
951 unique ID in its database, and create a new domain object that has the specified information. The new
952 IDs MAY return by Confirm message if the requester needs confirmation.

953



954

955

956 *Figure 3.6 Add transactions*

957

958 **Example: Document to add three Product Records**

```
959 <Document id="A-1" name="Product" action="Add">  
960 <Item id="001" name="Product-1"><Spec type="pps:color"><Char value="red"/></Spec></Item>  
961 <Item id="002" name="Product-2"><Spec type="pps:color"><Char value="red"/></Spec></Item>  
962 <Item id="003" name="Product-3"><Spec type="pps:color"><Char value="red"/></Spec></Item>  
963 </Document>
```

964

965 When *Condition* element is specified in a domain element, the *Property* element in the *Condition* element
 966 shows common property of all domain objects listed in the document. The following example is the same
 967 request compare to the previous example.

968

969 **Example:** Add document using a *Condition* element

```

970 <Document id="A-2" name="Product" action="Add" >
971 <Condition>
972   <Property name="pps:color"><Char value="red"/></Property>
973 </Condition>
974 <Item id="001" name="Product-1"/>
975 <Item id="002" name="Product-2"/>
976 <Item id="003" name="Product-3"/>
977 </Document>
  
```

978

979 The response to Add document can be done by sending a Confirm document that has primitive elements
 980 in its body. The primitive element represents the domain object that is successfully added, and SHOULD
 981 only have *id* attribute. The next example is the Confirm document as a result of the previous Add
 982 document.

983

984 **Example:** Confirm document as a response of an Add transaction

```

985 <Document id="B-1" name="Product" action="Confirm" >
986 <Item id="001" />
987 <Item id="002" />
988 <Item id="003" />
989 </Document>
  
```

990

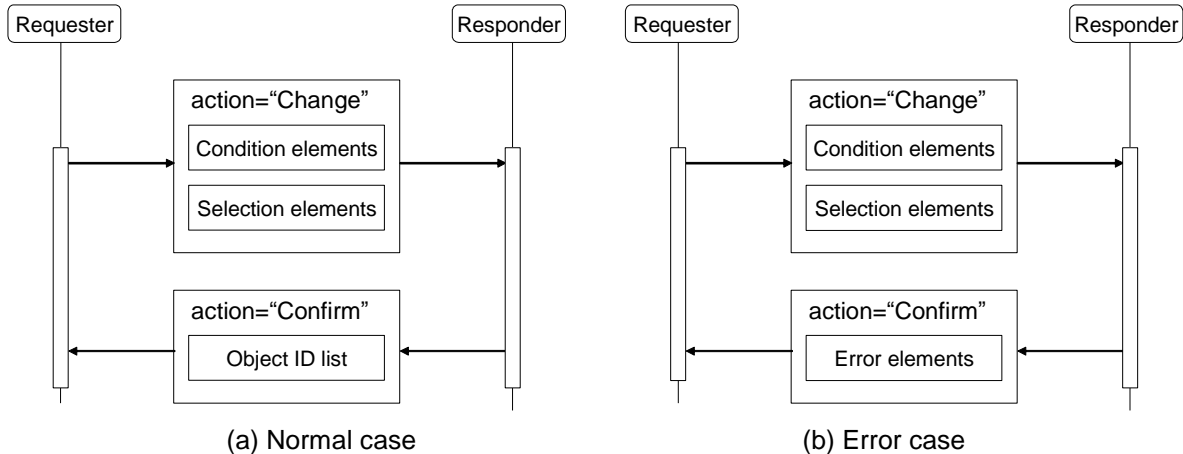
991 3.2.2 Change transaction

992 Change document requests to change the specified information of the specified domain objects that is in
 993 the database managed by the responder. In order to identify the target domain object, *Condition* element
 994 has any condition to select one or more than one domain objects.

995 After selecting the target domain object, Select element SHOULD represent the values of target
 996 properties to be changed. The values SHOULD be specified in the *Property* element in the *Selection*
 997 element.

998 All the selected domain objects depending on the *Condition* element SHOULD be applied to change in
 999 the same way. ID of domain objects SHOULD NOT be changed by this Change process.

1000



1001

1002

1003 *Figure 3.7 Change transactions*

1004

1005 In the database managed by the responder, a property type is either single or multiple. If the property
1006 type is single, the value requested to change is applied as a new value of the property. Otherwise, in the
1007 cases of multiple properties, the property of the domain object is inserted, updated or deleted depending
1008 on the type of the Change document.

1009 3.2.2.1 Insert property (Level 2 function)

1010 For the multiple primitives that have the same property name in the same object, an insert property
1011 document performs to add another property that has a new value. When *type* attribute of *Selection*
1012 element has "Insert" value, it shows that the properties in the *Selection* element are requested to insert.

1013

1014 **Example:** Add information of new level 10 as the latest stock value.

```
1015 <Document id="A-4" name="Product" action="Change" >  
1016 <Condition id="001"/>  
1017 <Selection type="Insert" >  
1018   <Property name="pps:stock"><Qty value="10"/></Property>  
1019 </Selection>  
1020 </Document>
```

1021

1022 3.2.2.2 Update property (Level 2 function)

1023 When the value of *type* attribute of *Selection* element is "Update", the properties in the *Selection* element
1024 are for updating the current properties in the owner's database. The target properties to be changed are
1025 selected by *Condition* elements which are defined under the *Selection* element.

1026 If the *Condition* elements select more than one property instances, all values of these selected instances
1027 are changed to the value specified in the *Property* element. If the *Condition* elements select no property
1028 instance, nothing happens for the message.

1029

1030 **Example:** Document requests to change the usage of A001-2 from 1 to 4.

```
1031 <Document id="A-5" name="Product" action="Change" >  
1032 <Condition id="A001"/>  
1033 <Selection type="Update" >  
1034   <Condition><Property name="pps:child"><Char value="A001-2"/></Property></Condition>  
1035   <Property name="pps:child-value"><Qty value="4"/></Property>  
1036 </Selection>  
1037 </Document>
```

1038

1039 **Example:** Initial status of the product data A001 that has A001-1, A001-2 and A001-3.

```
1040 <Document name="Item" id="A001">  
1041 <Compose type="pps:child" item="A001-1"><Qty value="1"/></Compose>  
1042 <Compose type="pps:child" item="A001-2"><Qty value="1"/></Compose>  
1043 <Compose type="pps:child" item="A001-3"><Qty value="1"/></Compose>  
1044 </Document>
```

1045

1046 **Example:** Revised status of the product data

```
1047 <Document name="Item" id="A001">  
1048 <Compose type="pps:child" item="A001-1"><Qty value="1"/></Compose>  
1049 <Compose type="pps:child" item="A001-2"><Qty value="4"/></Compose>  
1050 <Compose type="pps:child" item="A001-3"><Qty value="1"/></Compose>  
1051 </Document>
```

1052

1053 **3.2.2.3 Delete property (Level 2 function)**

1054 When a value of *type* attribute of *Selection* element is “Delete”, then it performs to delete particular
 1055 properties that are selected by *Condition* elements under the *Selection* element. *Condition* element is
 1056 necessary to select the target properties to be deleted.

1057 If the *Condition* elements select more than one property instances, all of these instances are deleted. If
 1058 the *Condition* elements select no property instance, nothing happens for the message.

1059
 1060 **Example:** Usage of “Machine-1” by the process “Proc-1” is requested to delete.

```

1061 <Document id="A-6" name="ProductionProcess" action="Change" >
1062 <Condition id="Proc-01"/>
1063 <Selection type="Delete">
1064 <Condition><Property name="pps:equipment"><Char value="Machine-
1065 1"/></Property></Condition>
1066 </Selection>
1067 </Document>
  
```

1068
 1069 **Example:** Delete all inventory records of the item “A001” that has a date before August 1st.

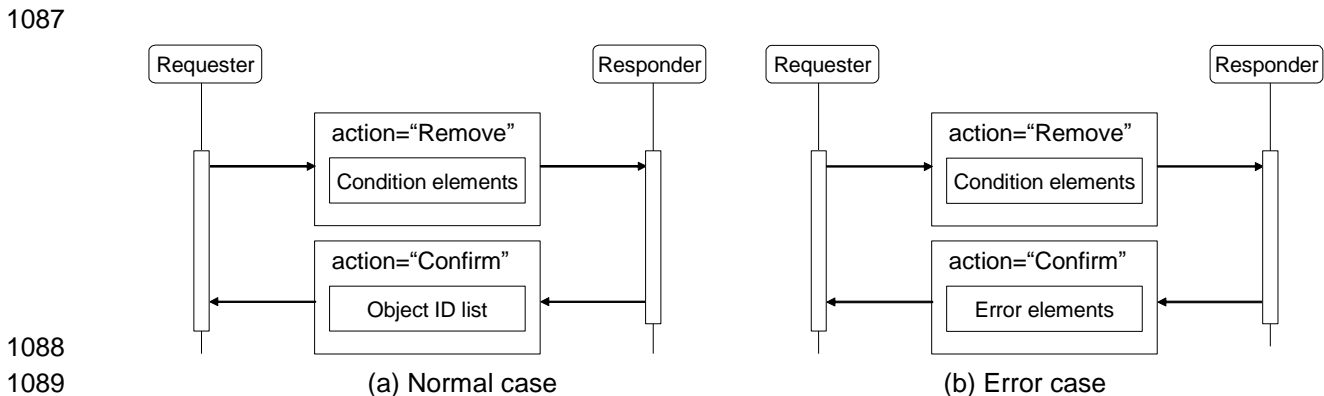
```

1070 <Document id="A-7" name="InventoryRecord" action="Change" >
1071 <Condition id="A001"/>
1072 <Selection type="delete">
1073 <Condition><Property name="pps:stock-date">
1074 <Time value="2006-08-01T00:00:00" condition="Max"/></Property>
1075 </Condition>
1076 </Selection>
1077 </Document>
  
```

1078
 1079 **3.2.3 Remove transaction**

1080 Remove document requests to delete the specified domain objects in the database managed by the
 1081 responder. The responder can decide either the request is accepted or rejected. If it is rejected, the
 1082 responder SHOULD send an error message, unless the confirm attribute is “Never”. Removing objects
 1083 means that the data in the owner’s database is actually deleted, or logically deleted such that only the
 1084 delete flag is marked on the object.

1085 The target domain objects to be removed are selected by specifying *Condition* elements that represent
 1086 the conditions of the target domain objects.



1088
 1089 *Figure 3.8 Remove transactions*

1090
 1091
 1092 **Example:** Document requesting that all the lot schedule objects of item “M001” are removed.

```

1093 <Document id="A-8" name="LotSchedule" action="Remove" >
1094 <Condition>
  
```

1095
1096
1097

```
<Property name="pps:item"><Char value="M001"/></Property>  
</Condition>  
</Document>
```

1098
1099

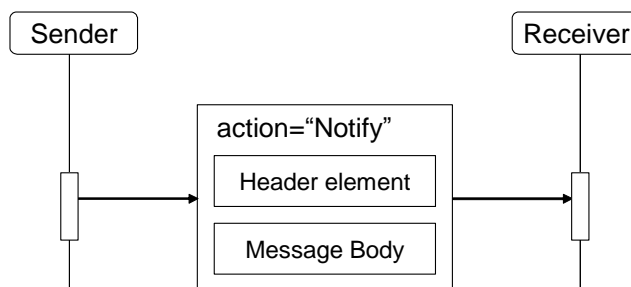
1100 3.3 Notify and Sync (NOTIFY and SYNC model)

1101 3.3.1 Notify transaction

1102 Notify document SHOULD have a value of "Notify" in the *action* attribute. The figure shows that
1103 transaction pattern of Notify document exchange. The sender of Notify document will not receive its
1104 response from the receiver.

1105 Notify document MAY be sent by the sender to any information users whom the sender decides as the
1106 destination of the message. If Notify document is caused by synchronization request specified by a Sync
1107 document received in advance, the message is sent when the corresponding event occurs. In Notify
1108 document for synchronization, the *event* attribute SHOULD show the event name.

1109



1110

1111 *Figure 3.9 Notify transactions*

1112

1113 Notify document SHOULD have a *Header* element that MAY have the number of domain objects and any
1114 aggregated information of objects. Domain objects, which are represented by primitive elements
1115 described in Section 2, MAY be described in the body of a Notify document.

1116

1117 **Example:** A Notify document shows reception of customer order 001 and its detailed items.

1118

```
<Document id="A-9" name="SalesOrder" action="Notify" >  
1119 <Header id="001" count="3" title="Order Form">  
1120 <Property type="Target" name="pps:party" display="C-Name"><Char value="K-  
1121 Inc."/></Property>  
1122 <Property type="Selection" name="pps:id" display="P/N"/>  
1123 <Property type="Selection" name="pps:name" display="NAME"/>  
1124 <Property type="Selection" name="pps:qty" display="QTY"/>  
1125 <Property type="Selection" calc="sum" name="pps:price" display="PRICE"><Qty  
1126 value="1200"/></Property>  
1127 </Header>  
1128 <Order id="001-1" item="Product-A1"><Spec type="pps:plan"><Qty  
1129 value="1"/></Spec></Order>  
1130 <Order id="001-2" item="Product-A2"><Spec type="pps:plan"><Qty  
1131 value="10"/></Spec></Order>  
1132 <Order id="001-3" item="Product-A3"><Spec type="pps:plan"><Qty  
1133 value="3"/></Spec></Order>  
1134 </Document>
```

1135

1136 3.3.2 Synchronizing process

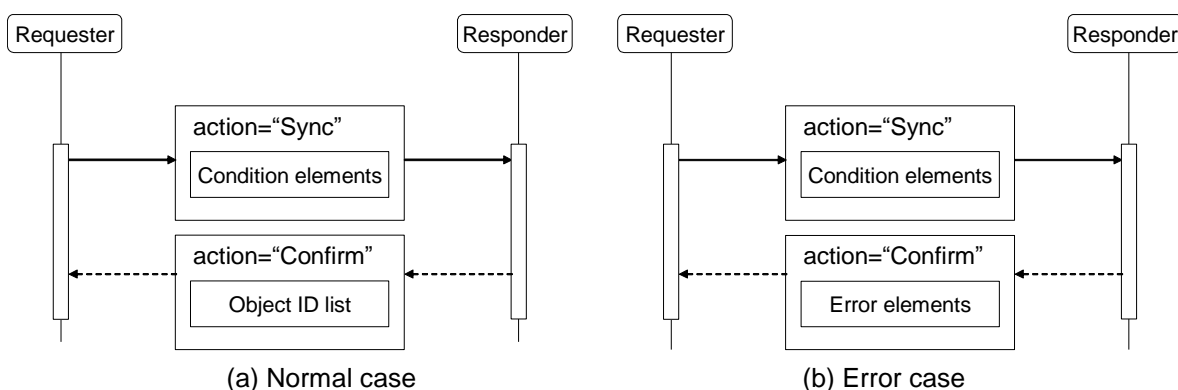
1137 In order to synchronize information of users with the information of the owner's database, the user needs
1138 to know the change of information at the time it occurs. The Sync transaction allows the user to request
1139 the information owner to notify the change of domain objects synchronously.

1140 If an information owner monitors particular property value of a domain object and tries to detect certain
1141 event occurrence such as data changes, the Sync document is used to establish a relationship of
1142 synchronization by requesting subscription of the event occurrence detected by the information owner.

1143 When a synchronization request specified using a Sync document is accepted by responder, e.g., the
1144 information owner, the responder SHOULD be ready to send a notification document by invoking another
1145 transaction when the corresponding event occurs. The notification documents are not included in the
1146 Sync transaction. Notification of change of the property value will be invoked as a different transaction
1147 independent from the Sync transaction.

1148 This model can be regarded as a publish-subscription model. The Sync document can be regarded as a
1149 subscription request message. If the responder has an additional subscription management module, then
1150 an application program can send a single Notify document to the module, which knows the subscribers
1151 and dispatch the message to all the members listed as a subscriber.

1152



1153

1154

1155 *Figure 3.10 Sync transaction*

1156

1157 All properties of a domain object MAY NOT be available to request for this synchronization service. In
1158 order to know the capability of application program and the list of event name that the application program
1159 can provide the service, an implementation profile defined in Section 4 SHOULD specify the information.

1160 According to the implementation profile specification format, the responder (information owner)
1161 determines the interval of monitoring cycle, size of difference to detect changes, range of value to detect
1162 event occurrence by minimum and maximum constraints, and so forth.

1163 When the value of the property is changed into the range defined by maximum and minimum constraints,
1164 the information owner SHOULD send the notification. The owner SHOULD NOT send a next notification
1165 of the event before the value will once be outside of the range.

1166 When the size of difference to detect changes is defined, any changes of the property value that is less
1167 than the size SHOULD be ignored.

1168 The changes during the monitoring cycle MAY be merged at the time of the next monitoring time.
1169 Therefore, changes during the cycle MAY NOT be detected by the requester.

1170 3.3.2.1 Sync document

1171 *Sync* document can represent a message to request synchronization of information. *Sync* document
1172 SHOULD specify a value "Sync" at *action* attribute of the element. *Sync* document SHOULD have an
1173 event name that has been defined in advance by the responder.

1174 *Sync* document MAY specify particular domain objects that have been managed by the responder at the
1175 time and is possible to monitor to detect the event. *Condition* element allows the requester to make
1176 request of synchronization for several domain objects by sending one *Sync* document.

1177 When there is no available event in the suggested domain object described by the event attribute and
1178 *Condition* elements, the responder SHOULD send a error information in *Confirm* document unless the
1179 request has “Never” value on the *confirm* attribute.

1180

1181 **Example:** To request notification when event “E01” occurs on any production order of item “A001”.

```
1182 <Document id="A-3" name="ProductionOrder" action="Sync" event="E01" >  
1183 <Condition>  
1184 <Property name="pps:item"><Char value="A001"/></Property>  
1185 </Condition>  
1186 </Document>
```

1187

1188 **Example:** The requester is registered in the subscription list of event “E01” on the three orders.

```
1189 <Document id="B-1" name="ProductionOrder" action="Confirm" event="E01" >  
1190 <Order id="1201"/>  
1191 <Order id="1204"/>  
1192 <Order id="1223"/>  
1193 </Document>
```

1194

1195 Once a *Sync* document is received without error, the synchronization request becomes effective until the
1196 responder will get a cancel request of the subscription, or the responder will stop the event detection
1197 process. In order to cancel the *Sync* request by requester, the requester SHOULD send a *Sync*
1198 document under a *Transaction* element that has *type* attribute with “Cancel” vale. When the responder
1199 receives cancelation of the *Sync* transaction, the responder SHOULD cancel the synchronization request
1200 corresponding to the transaction id. If the cancel request has new transaction id, then all transactions
1201 restricted by the specified event name and *Condition* element are canceled.

1202 3.3.2.2 Procedure of information owner

1203 Information owner, who has a capability of event monitoring and publishing services, MAY specify the
1204 available event information on the implementation profile described in Section 4. In accordance with the
1205 specification of the profile, the owner SHOULD perform event detection and publication.

1206 First, the information owner SHOULD monitor the actual value of the property that the owner decides to
1207 detect the event. In every monitoring cycle, the owner SHOULD determine whether the event occurs, that
1208 is, the value of the data is changed to satisfy all the conditions defined to the event. The conditions
1209 include minimum value, maximum value, and difference of change of the domain property.

1210 When the event occurs, the information owner SHOULD send a *Notify* document to all the members who
1211 are in the list of subscription. This is similar to publish-subscription mechanism, so the information owner
1212 MAY ask the publication process to a middle-ware information broker.

1213 The *Notify* document SHOULD have the event name at *event* attribute. The transaction id SHOULD be
1214 equal to the transaction id of the corresponding *Sync* document. The *Notify* document of this event
1215 occurrence SHOULD have the id of the domain object and the value of the property in the message body.

1216

1217 **Example:** Notify of event “E01” that shows a change of “production result” of production orders.

```
1218 <Document id="B-2" name="ProductionOrder" action="Notify" event="E01" >  
1219 <Order id="1204">  
1220 <Produce><Qty value="200"/></Produce>  
1221 </Order>  
1222 </Document>
```

1223

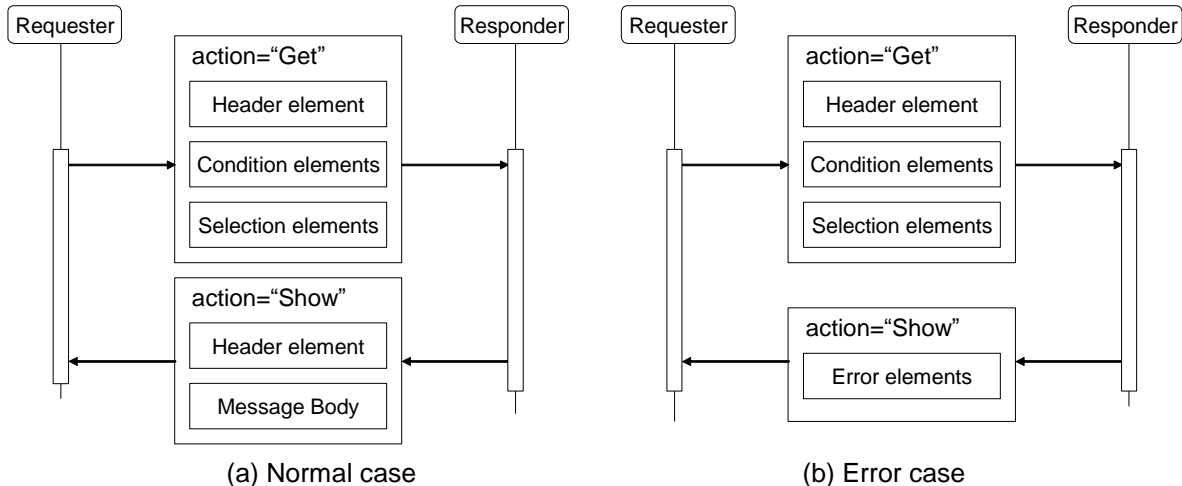
1224 3.4 Information Query (PULL model)

1225 Using a Get document, the requester MAY request particular information to the responder by describing
1226 the *Condition* elements that can select the target domain objects. The target objects can be described
1227 directly by IDs in *id* attribute, or any conditions of the domain objects using *Condition* elements.

1228 If no *Condition* element is specified in Get document, all domain objects that the responder manages in
1229 the database SHOULD be selected and shown in the content of the Show document.

1230 The responder who receives the Get document SHOULD process either responding corresponding
1231 domain objects, or refusing the request and setting error information in the Show document.

1232



1233

1234

1235 Figure 3.11 Get-Show transactions

1236

1237 3.4.1 Target domain objects

1238 3.4.1.1 Selection by object IDs

1239 The simplest way to select domain objects is describing IDs of the target objects in *Condition* elements. If
1240 the ID of the object is known, it can be specified as a value of *id* attribute of a *Condition* element. In this
1241 case, the *Condition* elements SHOULD be specified as many as the number of requested objects.

1242

1243 **Example:** Three objects that have "0001", "0005", "0013" as ID are requested.

```
1244 <Document id="A-2" name="Customer" action="Get" >  
1245 <Condition id="0001"/>  
1246 <Condition id="0005"/>  
1247 <Condition id="0013"/>  
1248 <Selection type="All"/>  
1249 </Document>
```

1250

1251 3.4.1.2 Selection by Property elements

1252 The second way to select domain objects is to specify *Property* elements in the *Condition* element under
1253 the *Document* element. The *Property* elements in this case represent condition of domain objects that
1254 SHOULD have the corresponding property. Each *Property* element shows the property name and its
1255 value, or range of value.

1256 If the data type of value is string, then the property shows that the *value* attribute should have the
1257 specified value.

1258 In order to select domain objects, the responder SHOULD evaluate the truth of the constraint described in
1259 the property, and if all the *Property* elements in the parent *Condition* element are satisfied, then the
1260 domain object SHOULD be selected.

1261

1262 **Example:** Products that have “white” as a value of color property are required.

```
1263 <Document id="A-3" name="Product" action="Get" >  
1264 <Condition>  
1265   <Property name="pps:color"><Char value="white" /></Property>  
1266 </Condition>  
1267 <Selection type="All"/>  
1268 </Document>
```

1269

1270 When a property specified in the *Condition* element is multiple, that is, the property can have many
1271 instances, the value of the corresponding *Property* element SHOULD meet at least one instance in the
1272 multiple property values.

1273

1274 **Example:** Any product items that has “A001” item in its parts list is required.

```
1275 <Document id="A-4" name="Product" action="Get" >  
1276 <Condition>  
1277   <Property name="pps:child"><Char value="A001"/></Property>  
1278 </Condition>  
1279 <Selection type="All"/>  
1280 </Document>
```

1281

1282 In order to select target objects, *Condition* element allows the requester to specify any range of property
1283 value. The range can be specified in *Property* element using *Qty*, *Char*, and *Time* element that has
1284 *condition* attribute. Available types of condition SHOULD include GE (greater than or equal), LE (less
1285 than or equal), GT (greater than), LT (less than), EQ (equal), NE (not equal).

1286

1287 **Example:** The document requests any products that the price is \$2,000 or higher.

```
1288 <Document id="A-5" name="Product" action="Get" >  
1289 <Condition>  
1290   <Property name="pps:price"><Qty value="2000" condition="GE"/></Property>  
1291 </Condition>  
1292 <Selection type="All"/>  
1293 </Document>
```

1294

1295 3.4.1.3 Disjunctive and conjunctive conditions

1296 When more than one *Property* elements are specified in a *Condition* element, it means that all conditions
1297 represented by the *Property* elements SHOULD be satisfied.

1298

1299 **Example:** Both A001 and A002 are the child items of the product.

```
1300 <Document "A-6" name="Product" action="Get" >  
1301 <Condition>  
1302   <Property name="pps:child"><Char value="A001"/></Property>  
1303   <Property name="pps:child"><Char value="A002"/></Property>  
1304 </Condition>  
1305 <Selection type="All"/>  
1306 </Document>
```

1307

1308 When there are more than one *Condition* elements in a document, these conditions are interpreted
1309 disjunctive, i.e., at least one condition SHOULD be satisfied.

1310

1311 **Example:** Compare to the previous example, the document shows a request of product data that has
1312 either A001 or A002 as a child part.

```
1313 <Document id="A-7" name="Product" action="Get" >  
1314 <Condition><Property name="pps:child"><Char value="A001"/></Property></Condition>  
1315 <Condition><Property name="pps:child"><Char value="A002"/></Property></Condition>  
1316 <Selection type="All"/>  
1317 </Document>
```

1318

1319 3.4.1.4 Selection by wildcard

1320 The third way to select target domain objects is to use wildcard in *Condition* element. To specify the
1321 required objects, *wildcard* attribute denotes the property name while the wildcard string is specified in the
1322 *value* attribute. The regular expressions [PCRE] are applied for interpreting the wildcard string.

1323 Wildcard specification SHOULD only apply to properties that have a value in string format.

1324

1325 **Example:** Request of customer orders that the destination address has any text of "Boston".

```
1326 <Document id="A-8" name="SalesOrder" action="Get" >  
1327 <Condition wildcard="pps:delivery" value="Boston"/>  
1328 <Selection type="All"/>  
1329 </Document>
```

1330

1331 3.4.2 Target domain property

1332 When the target domain objects are determined, *Get* document needs another specification for selecting
1333 properties in the domain objects to show the information detail. *Selection* element MAY be used for this
1334 purpose. The properties selected by *Selection* elements are included and corresponding values are
1335 described by the responder in the *Show* document.

1336 *Selection* element MAY represent ordering request/result of the objects in the response message, or
1337 calculating request/result of the values of the target objects.

1338 3.4.2.1 All available properties

1339 When the *type* attribute of *Selection* element has a value of "All", it SHOULD represent that all the
1340 possible properties are included in the *Show* document. The list of properties to return is decided by the
1341 responder.

1342 When value "Typical" is described in the *type* attribute, the typical properties of the domain object are
1343 selected by the responder. The list of typical properties is depending on the domain document. This list is
1344 defined by the responder according to the profile defined in Section 4.

1345

1346 **Example:** Request all the material information. All objects are selected with all possible properties.

```
1347 <Document id="A-9" name="ResourceCapacity" action="Get" >  
1348 <Selection type="All"/>  
1349 </Document>
```

1350

1351 3.4.2.2 Selecting domain property

1352 In order to specify the properties required in the selected objects, *Property* element in the *Selection*
1353 element is used. To select objects, name of property SHOULD be described in the *name* attribute of
1354 *Property* element in the *Get* document. Property name is defined in the application profile or the
1355 implementation profile.

1356
1357 **Example:** The objects in the responding document are required with properties of key, name and priority.

```
1358 <Document id="A-10" name="Party" action="Get" >  
1359 <Selection>  
1360 <Property name="pps:key"/>  
1361 <Property name="pps:name" />  
1362 <Property name="pps:priority" />  
1363 </Selection>  
1364 </Document>
```

1365
1366 When the property required has not been defined in the profile, Get document MAY request user-made
1367 properties by specifying its own texts following the prefix of "user:".

1368

1369 3.4.2.3 Sorting by property value (Level 2 function)

1370 Sorting request of the domain objects in the Show document can be described in *Property* element in
1371 *Selection* element. The *Property* element has *sort* attribute that MAY have a value of "Disc" or "Asc". The
1372 responder who receives this document SHOULD sort the domain objects by descending or ascending
1373 order, respectively.

1374 When there is more than one *Property* elements in the *Selection* element that has *sort* attribute, the first
1375 *Property* element is the highest priority of the sort procedure. If the values of the property of two objects in
1376 the responding domain objects are the same, then the second data value indicated by the next *Property*
1377 element are compared.

1378

1379 **Example:** Data request with sorting

```
1380 <Document id="A-12" name="Product" action="Get" >  
1381 <Selection>  
1382 <Property name="pps:parent" sort="Asc"/>  
1383 <Property name="pps:name" sort="Asc"/>  
1384 <Selection>  
1385 </Document>
```

1386

1387 **Example:** An example of response of the previous example

```
1388 <Document id="B-12" name="Product" action="Show" >  
1389 <Item name="bbb"><Compose type="pps:parent" item="A"/></Item>  
1390 <Item name="ccc"><Compose type="pps:parent" item="A"/></Item>  
1391 <Item name="ddd"><Compose type="pps:parent" item="A"/></Item>  
1392 <Item name="aaa"><Compose type="pps:parent" item="B"/></Item>  
1393 </Document>
```

1394

1395 3.4.2.4 Calculation of property value (Level 2 function)

1396 *Property* element in a *Selection* element MAY represent a request of calculation of property values that
1397 are selected by the *Get* document. In order to do this, *calc* attribute of *Property* element is used to select
1398 a calculation method. The value of *calc* attribute of *Property* element can take either "Sum", "Ave", "Max",
1399 "Min", and "Count" as a calculation function.

1400 The name of property that should be calculated MAY be described in *name* attribute of the *Property*
1401 element. Then, the values of the property SHOULD be calculated using the function describing at the *calc*
1402 attribute.

1403 In *Show* document or *Notify* document, the result of calculation is described in *Property* element in the
1404 *Header* element. Because *Show* and *Notify* element doesn't have *Selection* element, the result need to
1405 move from the *Selection* element in the *Get* document to the *Header* element.

1406 The responder who receives *Get* document SHOULD answer by calculating the target property value,
1407 and describes it at the corresponding *value* attribute of Qty, Char and Time element in the Property
1408 element depending on the data type.

1409

1410 **Example:** Requests to calculate summary of total price

```
1411 <Document id="A-13" name="SalesOrder" action="Get" >  
1412 <Selection>  
1413   <Property name="pps:price" calc="Sum"/>  
1414 </Selection>  
1415 <Selection type="All"/>  
1416 </Document>
```

1417

1418 **Example:** The corresponding response of the previous example

```
1419 <Document name="SalesOrder" id="B-13" action="Show" >  
1420 <Header count="3">  
1421   <Property name="pps:price" calc="Sum"><Qty value="2500"/></Property>  
1422 </Header>  
1423 <Order id="001" item="Product-1"><Price><Qty value="1000" unit="USD"/></Price></Order>  
1424 <Order id="004" item="Product-1"><Price><Qty value="1000" unit="USD"/></Price></Order>  
1425 <Order id="007" item="Product-1"><Price><Qty value="500" unit="USD"/></Price></Order>  
1426 </Document>
```

1427

1428 The response message to the calculation request has the calculation result in *Property* element in *Header*
1429 element. If the calculation method is "Count", then the result value is the number of corresponding domain
1430 objects in the database. In order to know the number of data before the detailed query execution, this
1431 calculation request MAY be send without *Selection* element that shows the property items in the *Show*
1432 document. In the case that "Count" value is specified in *calc* attribute, name attribute of *Property* element
1433 MAY NOT be specified.

1434

1435 **Example:** Request of counting the number of data

```
1436 <Document id="A-14" name="SalesOrder" action="Get" >  
1437 <Selection>  
1438   <Property calc="Count"/>  
1439 </Selection>  
1440 </Document>
```

1441

1442 **Example:** The answer of the request of counting the data

```
1443 <Document id="B-14" name="SalesOrder" action="Show" >  
1444 <Header>  
1445   <Property calc="Count"><Qty value="55"/></Property>  
1446 </Header>  
1447 </Document>
```

1448

1449 This value is similar to the value of *count* attribute in *Header* element. The value described in the count
1450 attribute represents the actual number of objects in the document, whereas the value in Property element
1451 shows the actual number in the database managed by the responder.

1452

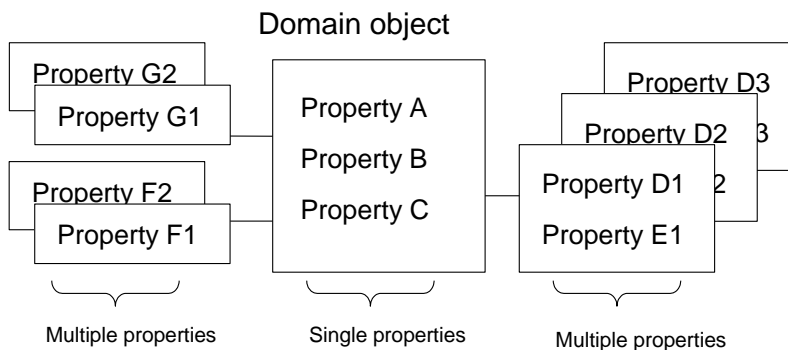
1453 3.4.3 Multiple property (Level 2 function)

1454 A *Document* element for a simple *Get* transaction has one *Selection* element which has several
1455 properties required by the sender. However, if the target domain object has a multiple property and some
1456 of its instances need to be selected, each multiple property SHOULD have corresponding *Selection*

1457 element. The *Selection* element for the multiple properties needs *Condition* element as its child element
1458 to represent conditions to select the instances.

1459 From a modeling perspective, a multiple property can be defined by attribute objects which are
1460 associated with or contained by the target domain object. The target domain object and attribute objects
1461 has one-to-many relations. Figure 3.12 shows that Property A, B, and C is a single property, while
1462 Property D to G are multiple properties. In this figure, it is important that Property D and E are on the
1463 same attribute object, and then any conditions for those two properties are applied in the same manner to
1464 select satisfied attribute objects.

1465



1466

1467 *Figure 3.12: Single property and Multiple property*

1468

1469 In accordance with this conceptual structure, a *Selection* element SHOULD be defined for each attribute
1470 class, i.e. type of attribute objects. For example, the case of the figure can have three different *Selection*
1471 elements. In the three *Selection* elements, one for the multiple properties has information of Property D
1472 and Property E at the same *Selection* element.

1473

1474 **Example:** A request of calendar information of a customer in April.

```
1475 <Document id="A-15" name="Customer" action="Get" >  
1476 <Condition id="001"/>  
1477 <Selection>  
1478   <Property name="pps:id" />  
1479   <Property name="pps:name"/>  
1480 </Selection>  
1481 <Selection>  
1482   <Property name="pps:calendar-date" />  
1483   <Property name="pps:calendar-value"/>  
1484   <Condition>  
1485     <Property name="pps:calendar-date">  
1486       <Time value="2006-04-01T00:00:00" condition="GE"/>  
1487     </Property>  
1488     <Property name="pps:calendar-date">  
1489       <Time value="2006-05-01T00:00:00" condition="LT"/>  
1490     </Property>  
1491   </Condition>  
1492 </Selection>  
1493 </Document>
```

1494

1495 **Example:** One possible answer to the previous document.

```
1496 <Document id="B-15" name="Customer" action="Show" >  
1497 <Party id="001">  
1498   <Capacity status="pps:holiday"><Time value="2006-04-01T00:00:00"/></Capacity>  
1499   <Capacity status="pps:work"><Time value="2006-04-02T00:00:00"/></Capacity>  
1500   <Capacity status="pps:work"><Time value="2006-04-03T00:00:00"/></Capacity>  
1501   ...  
1502   <Capacity status="pps:work"><Time value="2006-04-30T00:00:00"/></Capacity>  
1503 </Party>
```


1504 </Document>

1505
1506 When there is more than one *Selection* element in a transaction element, the first *Selection* element
1507 SHOULD NOT have *Condition* element. The *Selection* element that selects multiple properties SHOULD
1508 be specified at the second or later.

1509 3.4.4 Using Header element

1510 3.4.4.1 Inquiry by header element (Level 2 function)

1511 In a *Header* element of a Get document, brief inquiry information can be added independent from the
1512 main query mechanism provided by *Condition* and *Selection* elements. The brief inquiry mechanism is
1513 activated when *id* attribute of *Header* element in a *Get* document has an ID.

1514 The responder to this document SHOULD get the corresponding domain object which has the ID, and
1515 answer its property values required by *Primitive* elements of *Header* element in the Get document. The
1516 *Primitive* elements for the brief inquiry have *type* attribute with "Target" value, or the attribute doesn't have
1517 a value because "Target" is default value.

1518 The target object selected in this brief inquiry is basically in the same class of the domain objects, unless
1519 the *class* attribute of *Header* element has another name of domain object. When the *class* attribute is
1520 described with a name of another domain object, the corresponding information of the domain objects will
1521 be answered in the *Header* of the Show document.

1522 Multiple property MAY not be processed properly in this mechanism because the answer is formatted in
1523 single type properties. If a multiple property is selected in the *Header*, arbitrarily instance of the property
1524 is selected and described in the answer document.

1525
1526 **Example:** *Header* element for brief query has *id* attribute that is specified a name of the object.

```
1527 <Document id="A-16" name="Product" action="Get"  
1528 <Header id="001">  
1529   <Property type="Target" name="pps:name"/>  
1530 </Header>  
1531 </Document>
```

1532
1533 **Example:** An answer of the previous document

```
1534 <Document id="B-16" name="Product" action="Show" >  
1535 <Header id="001">  
1536   <Property type="Target" name="pps:name"><Char value="Product-A"/></Property>  
1537 </Header>  
1538 </Document>
```

1539

1540 3.4.4.2 Count of domain objects (Level 2 function)

1541 In Get document, *count* attribute of *Selection* element SHOULD represent the maximum number of
1542 objects described in the response message. If the value of the *count* attribute is 1 or more than 1, then
1543 the number described in the attribute restricts the size of the response message.

1544 When many domain objects are in the database, they can be retrieved separately by several Get
1545 documents. In such case, *offset* attribute of *Selection* element SHOULD be described as an offset
1546 number to skip the first objects while retrieving the domain objects.

1547 The offset request MAY be effective when a sort mechanism performed according to the value of *sort*
1548 attribute in *Property* element. If there is no description of sort, then the application MAY concern that the
1549 domain objects are sorted by the values of their IDs.

1550 The attribute of *count* and *offset* SHOULD NOT be specified if the *Selection* element is the second or
1551 later addressed in the *Document* element. In the corresponding Show document, the attribute of *count*
1552 and *offset* are specified in the *Header* element instead of *Selection* element.

1553

1554 **Example:** The following document requests customer order from #101 to #110.

```
1555 <Document id="A-17" name="SalesOrder" action="Get" >  
1556 <Selection offset="100" count="10"/>  
1557 <Property name="pps:id" sort="Desc"/>  
1558 </Selection>  
1559 </Document>
```

1560

1561 3.4.5 Show document

1562 3.4.5.1 Structure of Show document

1563 Show document has the same structure as the structure of Notify document. This document SHOULD
1564 have a value of "Show" at the *action* attribute.

1565 Show document SHOULD have header information by *Header* element, and if the Get document requests
1566 calculation by describing *calc* attribute of *Selection* elements, then the calculation results SHOULD be
1567 specified in *Header* element.

1568 Body of Show documents SHOULD have the content of the domain objects that corresponds to the
1569 request. The body MAY be empty if the corresponding object doesn't exist.

1570

1571 **Example:** The document of customer order #001 that has total amount and detailed item lists.

```
1572 <Document id="B-18" name="SalesOrder" action="Show" >  
1573 <Header id="001" count="3" title="OrderSheet">  
1574 <Property name="pps:party" display="CSTM"><Char value="K-Inc."/></Property>  
1575 <Property type="Selection" name="pps:id" display="PN"/>  
1576 <Property type="Selection" name="pps:name" display="NAME"/>  
1577 <Property type="Selection" name="pps:qty" display="QTY"/>  
1578 <Property type="Selection" calc="sum" name="pps:price" display="PRICE">  
1579 <Qty value="1200"/></Property>  
1580 </Header>  
1581 <Order id="001-1" item="Product-A1"><Qty value="1"/></Order>  
1582 <Order id="001-2" item="Product-A2"><Qty value="10"/></Order>  
1583 <Order id="001-3" item="Product-A3"><Qty value="3"/></Order>  
1584 </Document>
```

1585

1586 3.4.5.2 Header in Show document

1587 In Show documents, the number of domain objects listed in the body of the message is described as the
1588 value of *count* attribute of the *Header* element.

1589 *Property* elements described in the *Header* element consist of three types. First type is for properties of a
1590 header domain object requested by the Get document as a result of brief inquiry. All *Property* elements of
1591 this group SHOULD have a value "Target" at the *type* attribute or the attribute is not described. This
1592 property represents any value of the header object selected by *id* attribute of the *Header* element.

1593 The second type of *Property* elements has a value "Condition" at the *type* attribute. This property
1594 SHOULD represent that all domain objects listed in the body of the document has the same value
1595 described in the property. Application program who responses the Show document MAY describe the
1596 properties simply by duplicating the corresponding *Property* elements in *Condition* element in the Get
1597 document, because the property to be described can be regarded as a condition of the domain objects.

1598 The final group of properties comes from the *Selection* element of the Get document. The properties in
1599 this group SHOULD have a value "Selection" at the *type* attribute. These properties are basically a copy

1600 of *Property* elements of the *Selection* element in the Get document. If the *Selection* element in the Get
1601 document requests calculation, results are described in the *value* attribute of *Qty*, *Char* or *Time* sub-
1602 element of the *Property* element. In addition, a value of *display* attribute MAY be described for any texts
1603 in the header area for printing on a formatted sheet.

1604

1605 **Example:** A request to get product information of "A001" and its parts list.

```
1606 <Document id="A-19" name="Product" action="Get">  
1607 <Condition>  
1608   <Property name="pps:parent" value="A001"/>  
1609 </Condition>  
1610 <Selection>  
1611   <Property name="pps:id"/>  
1612   <Property name="pps:name"/>  
1613 </Selection>  
1614 <Header title="BillOfMaterials" id="A001" >  
1615   <Property name="pps:name"/>  
1616   <Property name="pps:price"/>  
1617   <Property name="pps:price-unit"/>  
1618 </Header>  
1619 </Document>
```

1620

1621 **Example:** The response to the previous Get document.

```
1622 <Document id="B-19" name="Product" action="Show">  
1623 <Header title="BillOfMaterials" id="A001" count="3">  
1624   <Property name="pps:name"><Char value="Product A001"/></Property>  
1625   <Property name="pps:price"><Qty value="2000"/></Property>  
1626   <Property name="pps:price-unit"><Char value="yen"/></Property>  
1627   <Property type="Condition" name="pps:parent"><Char value="A001"/></Property>  
1628   <Property type="Selection" name="pps:id"/>  
1629   <Property type="Selection" name="pps:name"/>  
1630 </Header>  
1631 <Item id="A001-01" name="Part A001-01"/>  
1632 <Item id="A001-02" name="Part A001-02"/>  
1633 <Item id="A001-03" name="Part A001-03"/>  
1634 </Document>
```

1635

1636

1637 3.5 XML Elements

1638 3.5.1 Message Structure

1639 Message is defined as unit information to send or receive by an application program at one time. A
1640 message that is exchanged between two parties SHOULD consist of one or more transaction elements or
1641 an implementation profile.

1642 The message content corresponds to any content in actual communication protocol such as SOAP, FTP
1643 and SMTP. Since this specification doesn't address on how to exchange messages in IP (Internet
1644 Protocol) level, data envelope mechanisms such as SOAP can be considered as well as a simple SMTP
1645 and file transfer mechanism.

1646 This information MUST be specified in the following XML schema.

1647

```
1648 <xsd:complexType name="MessageType">  
1649   <xsd:choice>  
1650     <xsd:element ref="ImplementProfile"/>  
1651     <xsd:element ref="Transaction" maxOccurs="unbounded"/>  
1652   </xsd:choice>  
1653   <xsd:attribute name="id" type="xsd:string" use="required"/>  
1654   <xsd:attribute name="sender" type="xsd:string"/>
```

1655
1656
1657
1658

```
<xsd:attribute name="security" type="xsd:string"/>  
<xsd:attribute name="create" type="xsd:dateTime"/>  
<xsd:attribute name="description" type="xsd:string"/>  
</xsd:complexType>
```

1659

- 1660 • *id* attribute SHOULD represent the identifier of the message. Every message SHOULD have a
1661 unique id in the scope of the sender or the requester.
- 1662 • *sender* attribute represents an identifier of the sender or requester of the message. This information
1663 is not for the low-level communication programs but for application programs.
- 1664 • *security* attribute represents a security text data such as pass words for authorization of the sender.
- 1665 • *create* attribute represents a date when the message is created.
- 1666 • *description* attribute represents any comments or descriptions.
- 1667
- 1668 • Elements under this messageType element SHOULD follow the sentences:
- 1669 • *ImplementProfile* element represents a request of implementation profile or answer of
1670 implementation profile defined in Section 4.
- 1671 • *Transaction* element represents transaction information to process in the responder.

1672

1673 In the case of representing XML format in messaging, the name of XML element can be described
1674 according to the following XML schema. In the case of describing in specific protocols such as SOAP, the
1675 payload body SHOULD be defined using MessageType.

1676

1677

```
<xsd:element name="Message" type="MessageType"/>
```

1678

1679 3.5.2 Transaction element

1680 A transaction element represents information of a transaction step. In the case where application need to
1681 commit several actions during transaction, and where it need to cancel and rollback the actions it has
1682 already processed, transaction element can control such operations.

1683 Transaction element SHOULD consist of zero or more than zero domain documents. When it has multiple
1684 documents, the first document in the content is the primary document in the transaction.

1685 This information MUST be specified in the following XML schema.

1686

1687

1688

1689

1690

1691

1692

1693

1694

1695

1696

1697

1698

1699

```
<xsd:element name="Transaction">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element ref="Document" minOccurs="0" maxOccurs="unbounded"/>  
    </xsd:sequence>  
    <xsd:attribute name="id" type="xsd:string" use="required"/>  
    <xsd:attribute name="type" type="xsd:string"/>  
    <xsd:attribute name="confirm" type="xsd:string"/>  
    <xsd:attribute name="connection" type="xsd:string"/>  
    <xsd:attribute name="create" type="xsd:dateTime"/>  
    <xsd:attribute name="description" type="xsd:string"/>  
  </xsd:complexType>  
</xsd:element>
```

1700

- 1701 • *id* attribute SHOULD represent the identifier of the transaction. Several transaction elements that
1702 belong to a transaction process SHOULD have same id value. For example, transaction elements in
1703 the same messaging model have the same id value. Re-sending depending on errors SHOULD have

- 1704 the same transaction id as the previous one. Every transaction process SHOULD have a unique id in
 1705 the scope of the sender or the requester.
- 1706 • *type* attribute represents transaction control type. “Start” SHOULD represent to start transaction,
 1707 while “Commit” SHOULD represent commitment and finalize the transaction. If the value is “Cancel”,
 1708 then it SHOULD represent that the transaction is canceled and the process stops.
 - 1709 • *confirm* attribute represents a confirmation request. The value of the attribute MUST be either
 1710 “Never”, “OnError”, or “Always”.
 - 1711 • *create* attribute represents a date when the transaction is created.
 - 1712 • *description* attribute represents any comments or descriptions.
- 1713
- 1714 • Elements under the transaction element SHOULD follow the sentences:
 - 1715 • *Document* element represents domain document to process in the responder.
- 1716

1717 3.5.3 Document element

1718 Domain document is information unit to perform actions by application programs. Domain document is
 1719 represented by document element. The specific list of domain documents which are necessary for
 1720 production planning and scheduling can be described by application profile defined in Section 4.

1721 This information MUST be specified in the following XML schema.

1722

```

1723 <xsd:element name="Document">
1724   <xsd:complexType>
1725     <xsd:sequence>
1726       <xsd:element ref="Error" minOccurs="0" maxOccurs="unbounded"/>
1727       <xsd:element ref="App" minOccurs="0"/>
1728       <xsd:element ref="Spec" minOccurs="0" maxOccurs="unbounded"/>
1729       <xsd:element ref="Condition" minOccurs="0" maxOccurs="unbounded"/>
1730       <xsd:element ref="Selection" minOccurs="0" maxOccurs="unbounded"/>
1731       <xsd:element ref="Header" minOccurs="0"/>
1732       <xsd:choice minOccurs="0">
1733         <xsd:element ref="Party" minOccurs="0" maxOccurs="unbounded"/>
1734         <xsd:element ref="Plan" minOccurs="0" maxOccurs="unbounded"/>
1735         <xsd:element ref="Order" minOccurs="0" maxOccurs="unbounded"/>
1736         <xsd:element ref="Item" minOccurs="0" maxOccurs="unbounded"/>
1737         <xsd:element ref="Resource" minOccurs="0" maxOccurs="unbounded"/>
1738         <xsd:element ref="Process" minOccurs="0" maxOccurs="unbounded"/>
1739         <xsd:element ref="Lot" minOccurs="0" maxOccurs="unbounded"/>
1740         <xsd:element ref="Task" minOccurs="0" maxOccurs="unbounded"/>
1741         <xsd:element ref="Operation" minOccurs="0" maxOccurs="unbounded"/>
1742       </xsd:choice>
1743     </xsd:sequence>
1744     <xsd:attribute name="id" type="xsd:string" use="required"/>
1745     <xsd:attribute name="name" type="xsd:string" use="required"/>
1746     <xsd:attribute name="ref" type="xsd:string"/>
1747     <xsd:attribute name="action" type="xsd:string"/>
1748     <xsd:attribute name="option" type="xsd:string"/>
1749     <xsd:attribute name="event" type="xsd:string"/>
1750     <xsd:attribute name="namespace" type="xsd:string"/>
1751     <xsd:attribute name="create" type="xsd:dateTime"/>
1752     <xsd:attribute name="description" type="xsd:string"/>
1753   </xsd:complexType>
1754 </xsd:element>
  
```

1755

- 1756 • *id* attribute SHOULD represent the identifier of the message. Every transaction message SHOULD
 1757 have a unique id in the scope of the sender or the requester.
- 1758 • *name* attribute represents name of domain document. The name SHOULD be selected from the list
 1759 in the application profile.

- 1760 • *ref* attribute represents the identifier of a primary message document or other document that is in the
1761 same transaction element, when the transaction element has more than one document.
- 1762 • *action* attribute represents the type of the message, where the types correspond to verbs information
1763 for the message. Values of the attribute is either “Add”, “Change”, “Remove”, “Confirm”, “Notify”,
1764 “Sync”, “Get”, or “Show”.
- 1765 • *option* attribute represents any optional information that may be interpreted by the receiver of the
1766 message.
- 1767 • *event* represents the identifier of event. When the document requests synchronization message, this
1768 value show the name of event the responder show in the profile. Notify document of the event also
1769 has the event name in this attribute.
- 1770 • *namespace* attribute represents namespace of the name of this document. When the implementation
1771 profile of the sender application supports more than one namespace, this attribute is required to
1772 identify the corresponding profile.
- 1773 • *create* attribute represents a date when the transaction document is created.
- 1774 • *description* attribute represents any comments or descriptions.

1775

1776 Elements under the transaction element SHOULD follow the sentences:

- 1777 • *Error* element represents error information.
- 1778 • *App* element represents any information for the application programs.
- 1779 • *Spec* element represents any particular specification of the document. This element is defined in
1780 Section 2.
- 1781 • *Condition* element represents any condition of selecting required domain objects.
- 1782 • *Selection* element represents any condition of selecting required properties of a domain object.
- 1783 • *Header* element represents information of the document independently defined from the domain
1784 objects.
- 1785 • *Party, Plan, Order, Item, Resource, Process, Lot, Task, or Operation* element represent domain
1786 objects. Different type of them SHOULD NOT be specified at the same parent *Document* element.

1787

1788 Action type that the document element has in its action attribute determines the structure of the element
1789 available to specify. The table below shows the combination matrix. Each column shows different
1790 document action type, while the row shows available elements in the document element. The blank cell
1791 represents the corresponding element SHOULD NOT be the child of the transaction element. “M” denotes
1792 that the corresponding element SHOULD be defined in the parent element. And “O” denotes optional where
1793 the element may described depending on the situation.

1794

1795 *Table 3.3 Structure of document element*

	Add	Change	Remove	Confirm	Confirm (Error)	Notify	Sync	Get	Show	Show (Error)
<i>Error</i> element					M					M
<i>App</i> element	O	O	O	O	O	O	O	O	O	O
<i>Condition</i> element	O	O	O				O	O		
<i>Selection</i> element		M						O		

Header element						M		O	M	O
Primitive element	M			M		M			M	

1796 3.5.4 Error element

1797 Error information SHOULD be specified in the error element under *Document* elements when one
 1798 application program needs to send the error results to the requester. The error elements MAY be
 1799 specified in Show documents and Confirm documents.

1800 The *Document* element SHOULD have one or more *Error* elements if the document is sent as error
 1801 information. The *Document* element SHOULD NOT have an *Error* element if the document is a normal
 1802 response in the messaging models.

1803 This information MUST be specified in the following XML schema. The XML documents generated by the
 1804 schema SHOULD be consistent with the following arguments.

1805

```

1806 <xsd:element name="Error">
1807   <xsd:complexType>
1808     <xsd:attribute name="id" type="xsd:string"/>
1809     <xsd:attribute name="ref" type="xsd:string"/>
1810     <xsd:attribute name="code" type="xsd:string"/>
1811     <xsd:attribute name="location" type="xsd:string"/>
1812     <xsd:attribute name="status" type="xsd:string"/>
1813     <xsd:attribute name="description" type="xsd:string"/>
1814   </xsd:complexType>
1815 </xsd:element>

```

1816

- 1817 • *id* attribute SHOULD represent identifier that application can identify the error data.
- 1818 • *ref* attribute represents the document id that has the errors.
- 1819 • *code* attribute represents unique identifier of the error categories. The error code MAY consist of
 1820 three digits. If the first digit is 0, then the code MAY represent as follows:
 - 1821 ➤ "000" represents "Unknown error".
 - 1822 ➤ "001" represents "Connection error".
 - 1823 ➤ "002" represents "Authorization error".
 - 1824 ➤ "003" represents "Application is not ready".
 - 1825 ➤ "004" represents "Message buffer is full".
 - 1826 ➤ "005" represents "Syntax error (communication)".
 - 1827 ➤ "006" represents "Syntax error (application logic)".
 - 1828 ➤ "007" represents "Requested task is not supported".
 - 1829 ➤ "008" represents "Requested task is denied".
 - 1830 ➤ "009" represents "No data object requested in the document".
 - 1831 ➤ "010" represents "Data object requested already exists".
 - 1832 ➤ "011" represents "Application error".
 - 1833 ➤ "012" represents "Abnormal exception".
- 1834 • *location* attribute represents the location of error texts.
- 1835 • *status* attribute represents a status. Values of this attribute SHOULD include:
 - 1836 ➤ "Error" represents that the document is error notification.
 - 1837 ➤ "Warning" represents that the document is warning.
- 1838 • *description* attribute represents any description of the error explanations.

1839 3.5.5 App element

1840 Application information MAY be used by application programs by their own ways. For this purpose, *App*
1841 element is defined. *App* element is extension area for application programs who may want to have their
1842 own information by using another name spaces. If the application programs within a messaging model
1843 can decide to have a new namespace, they have their own XML schema under the *App* element.

1844 This element MUST be consistent with the following XML schema.

1845

```
1846 <xsd:element name="App">  
1847   <xsd:complexType>  
1848     <xsd:sequence>  
1849       <xsd:any minOccurs="0" maxOccurs="unbounded"/>  
1850     </xsd:sequence>  
1851   </xsd:complexType>  
1852 </xsd:element>
```

1853

1854 3.5.6 Condition element

1855 *Condition* element SHOULD represent any condition to select domain objects or domain properties. The
1856 conditions can be defined by *Property* elements, which can represent value or range of property values.

1857 If there is more than one *Condition* element in the same XML element, then these conditions SHOULD be
1858 regarded disjunctive manner.

1859 This information MUST be specified in the following XML schema. The XML documents generated by the
1860 schema SHOULD be consistent with the following arguments.

1861

```
1862 <xsd:element name="Condition">  
1863   <xsd:complexType>  
1864     <xsd:sequence>  
1865       <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>  
1866     </xsd:sequence>  
1867     <xsd:attribute name="id" type="xsd:string"/>  
1868     <xsd:attribute name="wildcard" type="xsd:string"/>  
1869     <xsd:attribute name="value" type="xsd:string"/>  
1870     <xsd:attribute name="version" type="xsd:string"/>  
1871   </xsd:complexType>  
1872 </xsd:element>
```

1873

1874 • *Property* element represents any properties that restrict the target objects by describing a value or
1875 range of value.

1876

1877 • *id* attribute SHOULD represent the identifier of the target domain object. When the target object is
1878 known, then this value is specified instead of describing any other conditions.

1879 • *wildcard* attribute represents the name of property that is used to apply wildcard value. The wildcard
1880 text is specified in the *value* attribute.

1881 • *value* attribute represents the wildcard text for selecting the target domain objects. The text is
1882 interpreted by regular expression rules [PCRE].

1883 • *version* attribute represents version name of the target object. The format of version texts is
1884 managed in application programs. Values of this attribute MAY include:

- 1885 ➤ "Latest" --- the latest version object
- 1886 ➤ "Earliest" – the earliest version object
- 1887 ➤ any string that represent a version identifier

1888

1889 3.5.7 Selection element

1890 *Selection* element SHOULD represent information for appropriate properties to be selected in the all
1891 domain properties in the domain object. *Selection* elements are used in Get documents and Change
1892 documents.

1893 In Change documents, *Selection* element is used to select the property that the requester tries to change
1894 the value. In Get documents, *Selection* element is used to select the target properties to select in the
1895 Show document. If there is no *Select* element in Get document, then the corresponding Show document
1896 doesn't have any domain objects in its document body.

1897 When the target property of selection is multiple, then the parent Get document or Change document is
1898 required for each attribute object that the multiple property is defined.

1899 This information MUST be specified in the following XML schema. The XML documents generated by the
1900 schema SHOULD be consistent with the following arguments.

1901

```
1902 <xsd:element name="Selection">  
1903   <xsd:complexType>  
1904     <xsd:sequence>  
1905       <xsd:element ref="Condition" minOccurs="0" maxOccurs="unbounded"/>  
1906       <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>  
1907     </xsd:sequence>  
1908     <xsd:attribute name="type" type="xsd:string"/>  
1909     <xsd:attribute name="multiple" type="xsd:boolean"/>  
1910     <xsd:attribute name="count" type="xsd:int"/>  
1911     <xsd:attribute name="offset" type="xsd:int"/>  
1912   </xsd:complexType>  
1913 </xsd:element>
```

1914

- 1915 • *Condition* element represents any condition for selecting members of a multiple property, when the
1916 *multiple* attribute is "true". Change or Get document can restrict its target by this condition.
- 1917 • *Property* element represents any property required to describe in the target domain objects. In the
1918 case of Get document in PULL model, the corresponding information of this property is addressed in
1919 the body of the response document. More than one *Property* elements which represent multiple
1920 property SHOULD NOT be described in the same *Selection* element.
- 1921
- 1922 • *type* attribute represents the type of action after selecting the target properties. The available values
1923 are defined depending on the type of document.
 - 1924 ➤ "Insert" for Change document represents that the property value is inserted, this is default value.
1925 This value is not described in Get document.
 - 1926 ➤ "Update" for Change document represents that the property value is updated. This value is not
1927 described in Get document.
 - 1928 ➤ "Delete" for Change document represents that the property value is deleted. This value is not
1929 described in Get document.
 - 1930 ➤ "None" for Get document can represent that the target is specified by *Property* element. This is
1931 default value. This value is not described in Change document.
 - 1932 ➤ "Typical" for Get document can represent that the target property is typical set. This value is not
1933 described in Change document.
 - 1934 ➤ "All" for Get document can represent that the target property is all properties in the object. This
1935 value is not described in Change document.
- 1936 • *multiple* attribute for Get document shows whether the selected property is regarded as multiple or
1937 single one. If application profile or implementation profile shows that the property is single, then the
1938 selected property is regarded as single. No description of this attribute represents single property.

- 1939 • *count* attribute for Get document represents the maximum number of properties selected by the
- 1940 *Property* element for the domain object. This value is not described in Change document. This value
- 1941 is not be described for single property suggested by *multiple* attribute.
- 1942 • *offset* attribute for Get document represents the number of skipping the properties selected by the
- 1943 *Property* element for the domain object. This value is not described in Change document. This value
- 1944 is not described for single property suggested by *multiple* attribute.

3.5.8 Header element

Header element is used for representing header information in Show and Notify documents. The header information is described for any data depending on the document from an entire perspective. In Get document, *Header* element MAY be used to make brief inquiry of domain object that is not in the target of domain document. The *Header* element SHOULD be described in document elements.

This information MUST be specified in the following XML schema. The XML documents generated by the schema SHOULD be consistent with the following arguments.

```

<xsd:element name="Header">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="class" type="xsd:string"/>
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="count" type="xsd:int"/>
    <xsd:attribute name="offset" type="xsd:int"/>
  </xsd:complexType>
</xsd:element>

```

- 1966 • *Property* element represents any property of the target object in the header or any aggregation value of domain objects in the body of the document.
- 1967 • *id* attribute SHOULD represent ID of the target object that is shown in the header by describing its
- 1968 property in the “Property” element.
- 1969 • *class* attribute represents the target domain object that the header shows the information in its
- 1970 *Property* elements. If there is no class attribute, then it represents that the target domain object is
- 1971 those that the domain document refers to as default.
- 1972 • *title* attribute represents a title of the document.
- 1973 • *count* attribute represents the number of domain objects in the document. When this attribute is used
- 1974 in Notify document and Show document, the value equals to the number of object in the body of the
- 1975 document. In Get document, the value represents the maximum number of objects the receiver is
- 1976 expecting in the Show document.
- 1977 • *offset* attribute represents the offset number of data list. When the objects in the document are not all
- 1978 of the existing objects in the sender, the offset value shows the relative position of the first object on
- 1979 the document body in the whole objects. This attribute can be used in Get document as a request to
- 1980 offset the response data. In Notify and Show document, this value shows the offset number of the
- 1981 body.

3.5.9 Property element

Property element represents property information of domain objects under *Condition* element, *Selection* element and *Header* element. When *Condition* element has a *Property* element, it shows condition of

1989 selecting the domain objects. When *Selection* element has a *Property* element, it shows the target
 1990 property of changing or getting documents. When *Header* element has a *Property* element, it shows a
 1991 property of the header object or aggregation information of the body objects.
 1992 This information MUST be specified in the following XML schema. The XML documents generated by the
 1993 schema SHOULD be consistent with the following arguments.

1994

1995
 1996
 1997
 1998
 1999
 2000
 2001
 2002
 2003
 2004
 2005
 2006
 2007
 2008
 2009
 2010

```

<xsd:element name="Property">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="Qty" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Char" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Time" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="path" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string"/>
    <xsd:attribute name="sort" type="xsd:string"/>
    <xsd:attribute name="calc" type="xsd:string"/>
    <xsd:attribute name="display" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
  
```

2011

2012 • *Qty*, *Char*, and *Time* elements represent a value of the property. These elements are defined in
 2013 Section 2. When the property is described in *Condition* elements, constraint of property value MAY
 2014 be described, where the value attribute in *Qty*, *Char*, and *Time* element shows the value of
 2015 constraints, and condition attribute in *Qty*, *Char*, and *Time* element shows constraint type. Multiple
 2016 constraints under one property is regarded conjunctive.

2017

2018 • *type* attribute represents a type of property. This attribute is used only when the *Property* element is
 2019 defined under the *Header* element. The value of this attribute is one of the followings:
 2020 ➤ “Target” --- the property of the header target object,
 2021 ➤ “Condition” --- the condition data of the objects in the body. This data is copied from the property
 2022 data in the *Condition* element.
 2023 ➤ “Selection” --- the selection data of the properties of objects in the body. This data is copied from
 2024 the property data in the *Selection* element.

2025 • *name* attribute represents a name of property. The value of this attribute is the string that is defined
 2026 in the corresponding profile or a name of user-extended property whose name is starting with “user:”.

2027 • *path* attribute represents X-path string that shows the position of the data in the corresponding
 2028 primitive element. This attribute is required only if the value of the “name” attribute shows that the
 2029 property is user-extended property, because such path data is predefined in the profile for the others.

2030 • *value* attribute represents the value of property in *Selection* element and *Header* element. When this
 2031 attribute is described, then the value described in *Qty*, *Char* and *Time* SHOULD be ignored. When
 2032 the data type of this attribute is *Qty* or *Time*, then the value needs to be parsed to the corresponding
 2033 data type.

2034 • *sort* attribute represents that the objects in the body of this document are expected to be sorted by
 2035 ascending or descending order. For *Get* document, this attribute SHOULD be used in under
 2036 *Selection* element. For *Show* document and *Notify* document, this attribute SHOULD be specified in
 2037 *Header* element. If more than one *Property* element that has *sort* attribute are described in *Get*
 2038 document, these sort requests SHOULD be applied in the priority rule that the faster element
 2039 dominate the followers. This attribute SHOULD NOT use together with the *calc* attribute.

2040 ➤ “Asc” --- sort in ascending order,

2041 ➤ “Desc” --- sort in descending order.

- 2042
- 2043
- 2044
- 2045
- 2046
- 2047
- 2048
- 2049
- 2050
- 2051
- 2052
- 2053
- *calc* attribute represents that the property is expected to be calculated for the objects in the body of this document. For Get document, this attribute is used in *Selection* element. For Show document and Notify document, this attribute is described in *Header* element. This attribute does not use together with the *sort* attribute.
 - “Sum” --- summary of the value of properties of the target objects,
 - “Ave” --- average of the value of properties of the target objects,
 - “Max” --- maximum value of properties of the target objects,
 - “Min” --- minimum value of properties of the target objects,
 - “Count” --- the number of the target objects in the body.
 - *display* attribute represents the text string that can be shown in the header line for each primitive for explanation. This attribute is used only under the *Header* element.

2054 4 Profile Specifications

2055 4.1 Application profile Definitions

2056 4.1.1 General

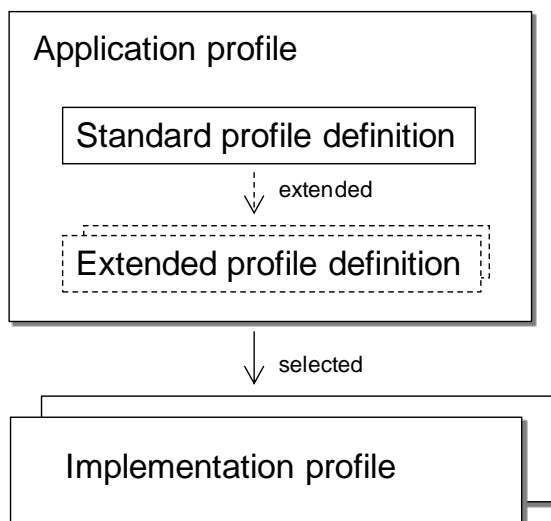
2057 Application profile definition is a set of specifications for all application programs that may be involved in
2058 the communication exchanging PPS transaction messages. Each application program may send and
2059 receive messages that consist of domain documents, domain objects and domain properties. The
2060 application profile definition provides all available domain documents, domain objects and domain
2061 primitives.

2062 Application programs can exchange their messages correctly when they understand the semantics of
2063 information in the message. In order to do this, application profile definition helps agreement of common
2064 usage and understanding of domain documents, domain objects and domain properties.

2065 Several application profile definitions can exist independently for the same problem domain. Two
2066 application programs cannot communicate each other if they don't refer a common application profile. In
2067 order to avoid such a situation, this specification provides an extension mechanism in which a standard
2068 profile definition can be extended to an extended profile definition for particular group in local domain.

2069 Figure 4.13 shows the structure of application profiles. Application profile is either a standard profile
2070 definition or an extended profile definition. Figure also shows that an implementation profile refers an
2071 application profile without regarding distinction of standard profile definition and extended profile
2072 definition.

2073



2074

2075 *Figure 4.13 Structure of profile specifications*

2076

2077 As an example of standard profile definition, PPS TC supports the PSLX profile [PROFILE] for this
2078 planning and scheduling domain. However, this specification only shows general rules and structures of a
2079 standard profile definition.

2080 4.1.2 Structure of profile definitions

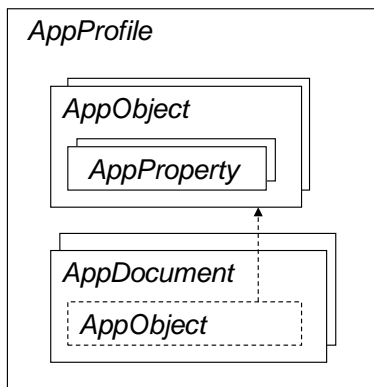
2081 Application profile SHOULD have a list of domain documents and a list of domain objects. In addition,
2082 application profile MAY have a list of enumerations, which shows available value set of a domain property
2083 of a domain object.

2084 Application profile definition SHOULD be described by *AppProfile* element defined in Section 4.3.1. This
2085 element SHOULD appear in the top level of the XML document.

2086 All candidates of domain documents, which may be used by any application program who sends or
2087 receives a message in the target domain, SHOULD be specified using *AppDocument* element under the
2088 *AppProfile* element.

2089 All domain objects, which are used in any domain document defined in *AppDocument* elements,
2090 SHOULD be specified in *AppObject* element under the *AppProfile* element. An *AppObject* has a list of
2091 properties that represent the characteristics of the object. Each property SHOULD be described in
2092 *AppProperty* under the *AppObject*.

2093



2094

2095 *Figure 4.14 Application Profile*

2096

2097 The structure of application profile is illustrated in Figure 4.14. Domain document represented by
2098 *AppDocument* has domain objects represented by *AppObject*. The domain objects that is listed in the
2099 same document SHOULD be the same class objects defined in one *AppObject* in the application profile.
2100 The application profile defines domain objects independent from domain documents, because the domain
2101 objects may be referred from several different kinds of domain documents.

2102

2103 **Example: Application profile definition**

```
2104 <AppProfile name="pps-profile" prefix="pps" namespace="http://www.oasis-  
2105 open.org/committees/pps/profile-1.0">  
2106   <AppObject name="Product" primitive="Item">  
2107     <AppProperty name="id" path="@id"/>  
2108     <AppProperty name="name" path="@name"/>  
2109     ...  
2110     <AppProperty name="Size" path="Spec[@type="size"]/@value"/>  
2111     <AppProperty name="Color" path="Spec[@type="color"]/@value"/>  
2112     ...  
2113   </AppObject>  
2114   ...  
2115   <AppDocument name="ProductRecord" object="Product"/>  
2116   <AppDocument name="ProductInventory" object="Product"/>  
2117   <AppDocument name="BillOfMaterials" object="Product"/>  
2118   <AppDocument name="BillOfResources" object="Product"/>  
2119   ...  
2120 </AppProfile>
```

2121

2122

2123 **4.1.3 Standard profile definitions**

2124 An application profile that does not have a base profile is a standard profile. Standard profile definition
2125 SHOULD be specified in consistent with the following rules:

- 2126 • Standard profile definition SHOULD have a name to identify the definition among all application
2127 programs in world-wide. Unique identifier such as URI is required.
- 2128 • The name of standard profile definition contains information of revision, and the revision of the
2129 definition SHOULD follow the rule defined in Section 4.1.5.
- 2130 • Standard profile definition SHOULD NOT have a base definition as a reference of other standard
2131 profile definitions.
- 2132 • Standard profile definition SHOULD be published among application programs and accessible by all
2133 the application programs in the problem domain via Internet by announcing the URL the application
2134 can download the document.
- 2135 • Standard profile definition SHOULD have the domain object in Table 4.4 or sub-class of Table 4.1
2136 domain objects. The domain objects SHOULD be represented by the primitive elements determined
2137 by the table.
- 2138 • Every domain object in a standard profile definition SHOULD have a domain property that shows
2139 identifier of the object. The domain property SHOULD be represented by id attribute of the primitive
2140 XML element in Table 4.1.

2141

2142 *Table 4.4 Domain objects required in standard profile definitions*

Object Name	XML Element	Description
Party	<i>Party</i>	Party such as customers and suppliers
Plan	<i>Plan</i>	Plan of production, capacity, inventory, etc.
Order	<i>Order</i>	Request of products and services
Item	<i>Item</i>	Items to produce or consume
Resource	<i>Resource</i>	Production resource such as machine and personnel
Process	<i>Process</i>	Production process
Lot	<i>Lot</i>	Actual lots produced in the plant
Task	<i>Task</i>	Actual tasks on certain resources
Operation	<i>Operation</i>	Actual operations in the plant

2143

2144 **4.1.4 Extended profile definitions**

2145 Standard profile definition MAY be extended by an extended profile definition. Extended profile definition
2146 MAY also be extended recursively. This is also represented by *AppProfile* element. Extended profile
2147 definitions SHOULD have a reference of a standard profile definition, which is the base of extension.

2148 Extended profile definition MAY add domain documents, domain objects and domain properties which
2149 have not been defined in the standard profile definition. Additional information of domain documents,
2150 domain objects and domain properties SHOULD be defined in the same way as the definition in standard
2151 profile definitions.

2152 Extended profile definitions MAY modify the domain documents, domain objects and domain properties
2153 addressed in the standard profile. In order to modify the definition, extended profile SHOULD describe
2154 new contents with the same identification name of the document, object or property.

2155 Extended profile definitions SHOULD NOT remove the domain documents, domain objects and domain
2156 properties addressed in the standard profile.

2157 Enumerations MAY be added or modified to the standard profile definition. When extended profile
2158 describes enumeration name which is in the standard profile, the candidates of the enumeration are

2159 replaced to those in the standard. Extended profile definitions SHOULD NOT remove any enumeration in
2160 the application profile.

2161

2162 **Example: Extended application profile**

```
2163 <AppProfile prefix="ex1" name="pps-profile-1.1" namespace="http://www.pslx.org/profile-  
2164 1" base="pps-profile-1.0">  
2165   <Enumeration name="groupType">  
2166     <EnumElement name="high" description="description of a"/>  
2167     <EnumElement name="low" description="description of b"/>  
2168   </Enumeration>  
2169   <AppObject name="Consumer">  
2170     <AppProperty name="group" path="Spec[type='pslx:group']/@value"  
2171     enumeration="groupType"/>  
2172   </AppObject>  
2173 </AppProfile>
```

2174

2175 Example shows an application profile extended from the standard profile. The new profile has additional
2176 enumeration named “groupType”, and then a new Consumer object is defined with a new property which
2177 has a name “group” and the additional enumeration type.

2178 4.1.5 Revision rule

2179 After an application profile definition has been created, many application programs are developed
2180 according to the profile definition. In accordance with the industrial experiences, the old definition may be
2181 required to modify for domain specific reasons in the application domain.

2182 Any application profile SHOULD NOT be changed without keeping the following rules after when the
2183 profile definition has been published. Otherwise, the new profile SHOULD have a new name that doesn't
2184 have any relation with the previous one.

2185 There are two revision levels. One is a revision that the system developers have to deal with the new
2186 specification and change if necessary. The other is editorial revision where the any program doesn't need
2187 to care in terms of interoperability. To inform the former cases, the name of profile SHOULD be changed
2188 by adding the revision numbers. For the latter cases, instead of changing the name of profile, the actual
2189 file name of the profile, specified at the *location* attribute in the *AppProfile* element SHOULD be changed.

2190 In order to represent the revision status in the profile name, there are two portions of digits in the name of
2191 profile definitions: major revision and minor revision. They are following the original identification name or
2192 the profile separated by dash “-” mark. The two portion is separated by the dot “.” character.

2193 When the major version increases, it:

- 2194 • SHOULD NOT change the name of the profile excepting the portion representing the revision status.
- 2195 • SHOULD NOT change the prefix and namespace in the attribute of *AppProfile* element.
- 2196 • SHOULD NOT change the domain object in *AppDocument* element.

2197 When the minor version increases, it:

- 2198 • SHOULD follow the rule of major version increasing,
- 2199 • SHOULD NOT change the domain properties in the domain objects.
- 2200 • SHOULD NOT change the enumeration definition in the *AppProfile* element.

2201

2202 4.2 Implementation profiles

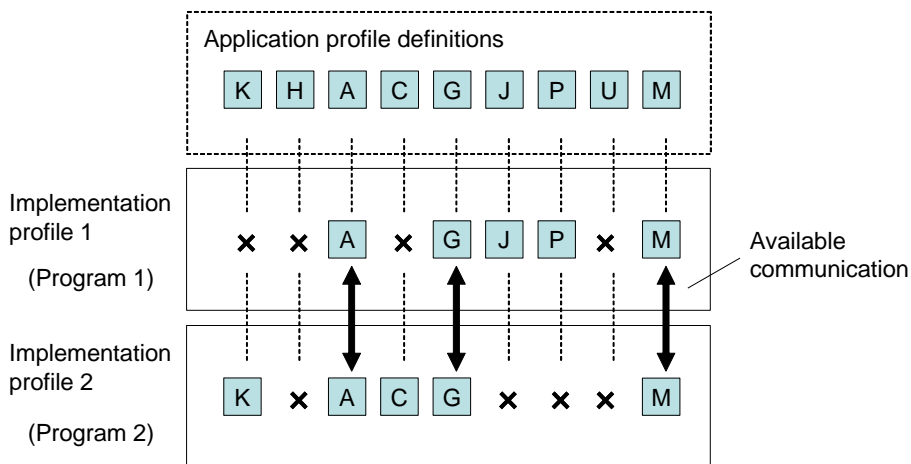
2203 4.2.1 General

2204 Application program may not have all capability in dealing with the domain documents, domain objects
2205 and domain properties defined in the application profile definitions. Implementation profiles are the

2206 selection of domain documents, domain objects and domain properties from application profile definitions
2207 by application programs depending on the capability of the program.

2208 When an application program tries to send a message to another application program, system integrator
2209 may need to confirm whether or not the receiving application program has capability to response the
2210 message. Then an implementation profile of an application program shows such capability to send or
2211 receive information.

2212



2213

2214 *Figure 4.15 Concept of communication availability between implementations*

2215

2216 Figure 4.15 explains a concept of communication availability between two application programs. Each
2217 application program that refers a same application profile has an implementation profile that has a list of
2218 items available to communicate, by selecting from the candidates defined in the application profile. Two
2219 application programs can exchange a message properly if the both implementations have the
2220 corresponding capability.

2221 An application program MAY have two or more than two implementation profiles each of which
2222 corresponding to different application profile definitions. An implementation profile SHOULD have a
2223 corresponding application profile definition.

2224 To confirm the capability of any application program, section 4.2.4 provides the method of how to get the
2225 information by receiving an implementation profile from the program.

2226 4.2.2 Structure of implementation profiles

2227 Implementation profiles defined for application programs SHOULD be described by *ImplementProfile*
2228 element in XML format. The information includes domain documents, domain objects and domain
2229 properties available to process by the application program. For each domain document, implementation
2230 level, which shows the application program have all functions or not in terms of transactions defined in
2231 Section 3, can be defined.

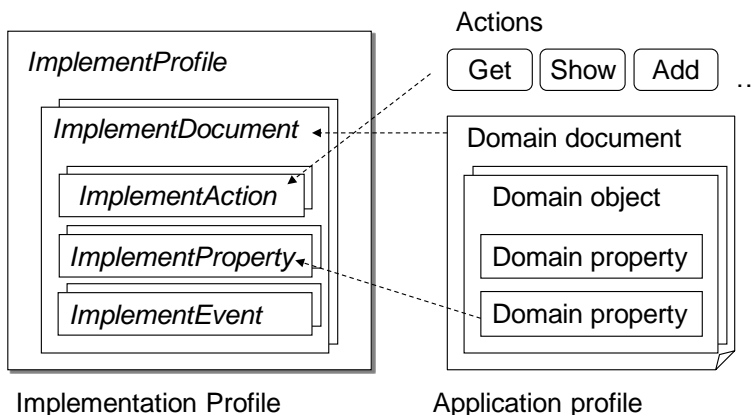
2232 Every implementation profile has a reference to a certain application profile. However, it doesn't show
2233 whether the application profile is a standard or extended. From the perspective of application programs,
2234 distinction between standard profile definition and extended profile definition has no sense.

2235 *ImplementProfile* element MAY be described under *Transaction* element defined in Section 3. Therefore,
2236 this can be send or receive through a PPS transaction process. Using Get and Show transactions, two
2237 application programs can exchange the implementation profile.

2238 An *ImplementProfile* element has *ImplementDocument* elements each of which represents availability for
2239 any domain document. An *ImplementDocument* element has *ImplementAction*, *ImplementProperty* and
2240 *ImplementEvent*.

2241 *ImplementAction* element represents information of implemented type of transaction such as Get, Show,
2242 Add, and so forth. *ImplementProperty* element represents implemented properties of the domain object.

2243 *ImplementEvent* represents any event definitions that the application program monitors properties and
 2244 publish notifications of event defined on the property. Figure 4.16 shows the structure of
 2245 *ImplementProfile*, *ImplementDocument*, *ImplementAction*, and *ImplementProperty* elements.
 2246



2247
 2248 *Figure 4.16 Structure of ImplementProfile*

2249
 2250 All domain documents represented by *ImplementProfile* SHOULD be in the list of the corresponding
 2251 application profile definition.
 2252 Domain documents in implementation profile SHOULD have a domain property if the property is defined
 2253 in the application profile as a primary key of the object or as a property that is always required.
 2254 The following example shows an implementation profile of an application program that communicates
 2255 with other program under an application profile. Then the implementation profile of the example is the
 2256 selection of the application profile representing domain documents, transaction types and domain
 2257 properties.

2258
 2259 **Example:** Implementation profile of a program for an application profile

```

2260 <ImplementProfile id="AP001" action="Notify">
2261   <ImplementDocument name="Product">
2262     <ImplementAction action="Get" level="1"/>
2263     <ImplementAction action="Show" level="1"/>
2264     <ImplementAction action="Add" level="2"/>
2265     <ImplementProperty name="id" title="Company ID"/>
2266     <ImplementProperty name="name" title="Company name"/>
2267   </ImplementDocument>
2268   <ImplementDocument name="ProductInventory">
2269     ...
2270   </ImplementDocument>
2271   ...
2272 </ImplementProfile>
  
```

2273
 2274 In accordance with the implementation profile, the application program sends or receives a message that
 2275 SHOULD have a domain document listed in the implementation profile. The domain properties in the
 2276 object SHOULD be one of the domain properties defined in the application profile.

2277
 2278 **Example:** A message created on the implementation profile

```

2279 <Document name="Product" id="001" action="Get"
2280   namespace="http://www.oasis-open.org/committees/pps/profile-1.0">
2281   <Condition>
2282     <Property name="pps:name" value="MX-001"/>
2283     <Property name="pps:color" value="white"/>
2284   </Condition>
  
```

```
2285 <Selection type="All"/>
2286 </Document>
```

2287
2288 Above example shows a message of a Get document created by an application program. The properties
2289 referred to as "name" and "color" are specified in this message. The properties are defined in the
2290 implementation profile as well as the application profile. The prefix "pps" and colon mark are added at the
2291 front of the name to notify that the name is defined in the profile.

2292 4.2.3 Level of implementation

2293 Domain documents can be sent or received by application programs in any types of action including Add,
2294 Change, Remove, Get, Show, Notify and Sync. These actions are prescribed in Section 3. Level of
2295 implementation represents whether or not the functions prescribed in Section 3 are fully implemented or
2296 partially implemented

2297 The certain level of Partial implementation is defined in Section 3 depending on the type of transaction.
2298 When the application program informs Partial implementation, it SHOULD have full capability of functions
2299 defined in the partial implementation in Section 3.

2300 An application program MAY define a level of implementation for each pair of document and transaction
2301 type for each application profile definition.

2302 4.2.4 Profile inquiry

2303 All application programs SHOULD send implementation profile as a Show transaction message or Notify
2304 transaction message. Application programs SHOULD have capability to response implementation profile
2305 as Show message when it receives an *ImplementProfile* inquiry in a form of Get message.

2306 When responding to the Get message of implementation profile in PULL model, the program SHOULD
2307 send corresponding Show message that is made of *ImplementProfile* element or *Error* element.

2308 This capability of implement profile inquiry SHOULD NOT be in the available list of *ImplementProfile* by
2309 itself. Since any *Condition* and *Selection* element cannot be described in *ImplementProfile*, the inquiry of
2310 implementation profile can only request all the information of implement profiles.

2311
2312 **Example:** Inquiry of implementation profile for PPS standard profile definition

```
2313 <Message id="A01" sender="A">
2314 <ImplementProfile action="Get" />
2315 </Message>
```

2316
2317 **Example:** Answer of the inquiry in above example

```
2318 <Message id="B01" sender="B">
2319 <ImplementProfile id="B01" action="Show" >
2320 <ImplementDocument name="Supplier">
2321 <ImplementAction action="Get" level="1"/>
2322 <ImplementAction action="Add"/>
2323 <ImplementProperty name="id" display="NO"/>
2324 <ImplementProperty name="name" display="NAME"/>
2325 ...
2326 </ImplementDocument>
2327
2328 </ImplementProfile >
2329 </Message>
```

2330
2331 Examples are the request of implementation profile and its response. By the message in the first example
2332 , the responder needs to answer its capability on the application profiles.

2333 4.3 XML Elements

2334 4.3.1 AppProfile Element

2335 *AppProfile* element SHOULD represent an application profile. Standard application profile and extended
2336 application profile are both represented by this element. This is a top level element in an application
2337 profile, and has *Enumeration* element, *AppObject* element, and *AppDocument* element.

2338 This information SHOULD be specified in the following XML schema. The XML documents generated by
2339 the schema SHOULD be consistent with the following arguments.

2340

```
2341 <xsd:element name="AppProfile">  
2342   <xsd:complexType>  
2343     <xsd:sequence>  
2344       <xsd:element ref="Enumeration" minOccurs="0" maxOccurs="unbounded"/>  
2345       <xsd:element ref="AppObject" minOccurs="0" maxOccurs="unbounded"/>  
2346       <xsd:element ref="AppDocument" minOccurs="0" maxOccurs="unbounded"/>  
2347     </xsd:sequence>  
2348     <xsd:attribute name="name" type="xsd:string" use="required"/>  
2349     <xsd:attribute name="base" type="xsd:string"/>  
2350     <xsd:attribute name="location" type="xsd:string"/>  
2351     <xsd:attribute name="prefix" type="xsd:string"/>  
2352     <xsd:attribute name="namespace" type="xsd:string"/>  
2353     <xsd:attribute name="create" type="xsd:string"/>  
2354     <xsd:attribute name="description" type="xsd:string"/>  
2355   </xsd:complexType>  
2356 </xsd:element>
```

2357

- 2358 • *Enumeration* element SHOULD represent any enumeration type that is used as a special type of
2359 properties.
- 2360 • *AppObject* element SHOULD represent any domain objects used in the domain documents defined in
2361 this profile.
- 2362 • *AppDocument* element SHOULD represent any domain documents that the applications may send or
2363 receive on this profile.
- 2364
- 2365 • *name* attribute SHOULD represent the name of this application profile. The name SHOULD be unique
2366 in the namespace. This attribute is REQUIRED.
- 2367 • *base* attribute SHOULD represent the base application profile when this profile is an extended
2368 application profile.
- 2369 • *location* attribute SHOULD represent the location where the profile can be downloaded via Internet.
- 2370 • *prefix* attribute SHOULD represent the prefix text that is added in the name of values that are
2371 qualified by this profile.
- 2372 • *namespace* attribute SHOULD represent the namespace when this profile is used in a specific
2373 namespace.
- 2374 • *create* attribute SHOULD represent the date of creation of the profile
- 2375 • *description* attribute SHOULD represent any description related to this profile.

2376 4.3.2 AppDocument Element

2377 *AppDocument* element SHOULD represent a domain document that is contained in a message of any
2378 transactions. All domain documents that may appear in messages SHOULD be described in
2379 *AppApplication* element that corresponds to an application profile.

2380 This information SHOULD be specified in the following XML schema. The XML documents generated by
2381 the schema SHOULD be consistent with the following arguments.

2382

2383
2384
2385
2386
2387
2388
2389
2390

```
<xsd:element name="AppDocument">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="object" type="xsd:string"/>
    <xsd:attribute name="category" type="xsd:string"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

2391

- 2392 • *name* attribute SHOULD represent the name of the domain document. The name SHOULD be unique
2393 in the namespace to identify the type of the document. This attribute is REQUIRED.
- 2394 • *object* attribute SHOULD represent the name of domain object that the document MAY have in the
2395 body as its content. One document SHOULD have one kind of domain object. All objects referred by
2396 this attribute SHOULD be defined in the same application profile or base application profile. This
2397 attribute is REQUIRED.
- 2398 • *category* attribute SHOULD represent any category of the domain document. This information is used
2399 for making any group by categorizing various documents. Same group documents have same value
2400 for this attribute. This attribute is OPTIONAL.
- 2401 • *description* attribute SHOULD represent any description of the domain document. Any comments and
2402 additional information of the document may be specified there. This attribute is OPTIONAL.

2403 4.3.3 AppObject Element

2404 *AppObject* element SHOULD represent a domain object corresponding to any actual object in the target
2405 problem domain. All domain objects that are referred to from domain documents in the application profile
2406 SHOULD be described in the *AppObject* element.

2407 This information SHOULD be specified in the following XML schema. The XML documents generated by
2408 the schema SHOULD be consistent with the following arguments.

2409

2410
2411
2412
2413
2414
2415
2416
2417
2418
2419

```
<xsd:element name="AppObject">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="AppProperty" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="primitive" type="xsd:string" use="required"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

2420

- 2421 • *AppProperty* element SHOULD represent a property that may be described in the domain objects of
2422 the application profile definition. All possible properties SHOULD be described in the domain object
2423 represented by *AppObject*.
- 2424
- 2425 • *name* attribute SHOULD represent the name of the object. The name SHOULD be unique under the
2426 application profile definition in the selected namespace. This attribute is REQUIRED.
- 2427 • *primitive* attribute SHOULD represent a primitive element name selected from the primitive element
2428 list defined in Section 2. Since every domain object is a subclass of one in the primitive objects, all
2429 *AppObject* elements SHOULD have a primitive attribute. This attribute is REQUIRED.
- 2430 • *description* attribute SHOULD represent any description of the domain object. This attribute is
2431 OPTIONAL.

2432 4.3.4 AppProperty Element

2433 *AppProperty* element SHOULD represent a domain property of a domain object. All properties that may
2434 be defined to represent the characteristics of the domain object SHOULD be described under the
2435 *AppObject* corresponding to the domain object.

2436 This information SHOULD be specified in the following XML schema. The XML documents generated by
2437 the schema SHOULD be consistent with the following arguments.

2438

```
2439 <xsd:element name="AppProperty">  
2440   <xsd:complexType>  
2441     <xsd:attribute name="name" type="xsd:string"/>  
2442     <xsd:attribute name="path" type="xsd:string"/>  
2443     <xsd:attribute name="multiple" type="xsd:string"/>  
2444     <xsd:attribute name="key" type="xsd:string"/>  
2445     <xsd:attribute name="enumeration" type="xsd:string"/>  
2446     <xsd:attribute name="dataType" type="xsd:string"/>  
2447     <xsd:attribute name="use" type="xsd:string"/>  
2448     <xsd:attribute name="description" type="xsd:string"/>  
2449   </xsd:complexType>  
2450 </xsd:element>
```

- 2451
- 2452 • *name* attribute SHOULD represent the name of the property. The name SHOULD be unique in the
2453 domain object defined by *AppObject* to identify the property. This attribute is REQUIRED.
 - 2454 • *path* attribute SHOULD represent the location of the attribute data in the primitive XML description
2455 defined in Section 2. The specification of the path SHOULD conform to [PATH]. If the profile is a
2456 standard application profile, this attribute is REQUIRED, and otherwise OPTIONAL.
 - 2457 • *multiple* attribute SHOULD represent whether the property can have multiple values or not. If the
2458 value of this attribute is positive integer or “Unbounded”, actual message described by Section 2
2459 specification can have corresponding number of values for this property. This attribute is OPTIONAL.
 - 2460 • *key* attribute SHOULD represent whether or not this property is primary key of the domain object to
2461 identify the target object from the instances in the database. If the value is “True”, then this property is
2462 primary key. Primary key SHOULD NOT defined more than one in the same domain object.
 - 2463 • *enumeration* attribute SHOULD represent the name of enumeration type when the property has a
2464 value in the enumeration list. The name of enumeration type SHOULD be specified in *Enumeration*
2465 elements in the same application profile definition. This attribute is OPTIONAL.
 - 2466 • *dataType* attribute SHOULD represent the data type of the property. The value of this attribute
2467 SHOULD be “Qty”, “Char” or “Time”. The data type described in the attribute SHOULD be the same
2468 as the data type of attribute on the body elements identified by the *path* attribute.
 - 2469 • *use* attribute SHOULD represent that the property is mandatory for any implementation, if the value of
2470 this attribute is “Required”.
 - 2471 • *description* attribute SHOULD represent any description of the domain property. This attribute is
2472 OPTIONAL.

2473 4.3.5 Enumeration Element

2474 *Enumeration* element SHOULD represent an enumeration type that has several items in a list format. If a
2475 property of a domain object has the enumeration type, then the property SHOULD have one of any items
2476 in the enumeration list.

2477 Enumeration type is independent from any domain object in the application profile definition. Therefore,
2478 several different domain objects MAY have different properties that has the same enumeration type.

2479 This information SHOULD be specified in the following XML schema. The XML documents generated by
2480 the schema SHOULD be consistent with the following arguments.

2481

2482
2483
2484
2485
2486
2487
2488
2489
2490

```
<xsd:element name="Enumeration">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="EnumElement" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

2491

2492 • *EnumElement* element SHOULD represent an item of the list that the enumeration type has as
2493 candidates of property value.

2494

2495 • *name* attribute SHOULD represent a name of this enumeration type. The name SHOULD be unique
2496 in the application profile definition. This attribute is REQUIRED.

2497 • *description* attribute SHOULD represent any description of the enumeration type. This attribute is
2498 OPTIONAL.

2499 4.3.6 EnumElement Element

2500 *EnumElement* element SHOULD represent an item of enumeration list. A property that is defined with the
2501 enumeration type SHOULD select one of the items from the enumeration list.

2502 This information SHOULD be specified in the following XML schema. The XML documents generated by
2503 the schema SHOULD be consistent with the following arguments.

2504

2505
2506
2507
2508
2509
2510
2511
2512

```
<xsd:element name="EnumElement">
  <xsd:complexType>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
    <xsd:attribute name="primary" type="xsd:boolean"/>
    <xsd:attribute name="alias" type="xsd:int"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

2513

2514 • *value* attribute SHOULD represent value texts that can be selected from the enumeration list. The
2515 value SHOULD be unique in the value list of the enumeration type. This attribute is REQUIRED.

2516 • *primary* attribute SHOULD represent the primary item in the enumeration list. Only the primary
2517 attribute SHOULD have "True" value for this attribute. No more than one item in the item list SHOULD
2518 have "true" value. This attribute is OPTIONAL, and the default value is "False".

2519 • *alias* attribute SHOULD represent a numerical value instead of the text value specified in the *value*
2520 attribute. The value SHOULD be unique integer among the items in the enumeration type.

2521 • *description* attribute SHOULD represent any description of the enumeration element. This attribute is
2522 OPTIONAL.

2523 4.3.7 ImplementProfile Element

2524 *ImplementProfile* element SHOULD represent an implementation profile for an application program.

2525 *ImplementProfile* SHOULD be defined for each application program what the application program
2526 supports. This information MAY be sent by the application program and received by the party who wants
2527 to know the capability of the application program. Therefore, in order to make transactions, some
2528 attributes and sub-elements are the same as the attributes of Document element defined in Section 3.

2529 This information SHOULD be specified in the following XML schema. The XML documents generated by
2530 the schema SHOULD be consistent with the following arguments.

2531

2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548

```
<xsd:element name="ImplementProfile">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Error" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="App" minOccurs="0"/>
      <xsd:element ref="ImplementDocument" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="action" type="xsd:string"/>
    <xsd:attribute name="profile" type="xsd:string"/>
    <xsd:attribute name="location" type="xsd:string"/>
    <xsd:attribute name="namespace" type="xsd:string"/>
    <xsd:attribute name="create" type="xsd:dateTime"/>
    <xsd:attribute name="description" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

2549

2550
2551
2552

- *Error* element SHOULD represent error information, when any errors occur during the transaction of message exchange of this implementation profile. The specification of this element is defined in Section 3.

2553
2554
2555
2556

- *App* element SHOULD represent any information for the application program concerning the transaction of profile exchange. The use of this element SHOULD be consistent with all cases of transactions while the other messages are exchanged. The specification of this element is defined in Section 3.

2557
2558
2559

- *ImplementDocument* element SHOULD represent a domain document that the application program may send or receive. All available documents in the application profile SHOULD be listed using this element.

2560

2561
2562
2563
2564
2565

- *id* attribute SHOULD represent identifier of the application program. The *id* SHOULD be unique in all application programs that can be accessed in the network. In order to guarantee the uniqueness, system integrator must assign the unique number and manages it in the network configuration. This *id* is the same as the sender name when the application will send a message. This attribute is REQUIRED.

2566
2567

- *name* attribute SHOULD represent a name that the application program shows its name for an explanation by natural texts. This attribute is OPTIONAL

2568
2569
2570
2571

- *action* attribute SHOULD represent a name of action during transaction models defined in Section 3. The value of this attribute SHOULD be “Notify”, “Get” or “Show”. When the element is created as a message for exchange, this attribute is REQUIRED. Otherwise, such as for a XML document file, this attribute is OPTIONAL.

2572
2573

- *profile* attribute SHOULD represent the name of application profile that this implementation profile is referring to select the available part in the definition. This attribute is OPTIONAL.

2574
2575

- *location* attribute SHOULD represent the location of the application profile to get the actual file by the party who want to know the content of the application profile. This attribute is OPTIONAL.

2576
2577

- *namespace* attribute SHOULD represent the namespace of the application profile. This attribute is necessary to identify the profile in world-wide basis. This attribute is OPTIONAL.

2578
2579

- *create* attribute SHOULD represent the date of creation of the implementation profile. This attribute is OPTIONAL.

2580
2581

- *description* attribute SHOULD represent any description of the implementation profile. This attribute is OPTIONAL.

2582

2583 4.3.8 ImplementDocument Element

2584 *ImplementDocument* element SHOULD represent a domain document selected from the application
2585 profile. All available domain documents SHOULD be listed by this element. Available domain documents
2586 MAY be defined for each application profile that the program can support.

2587 This information SHOULD be specified in the following XML schema. The XML documents generated by
2588 the schema SHOULD be consistent with the following arguments.

2589

```
2590 <xsd:element name="ImplementDocument">  
2591   <xsd:complexType>  
2592     <xsd:sequence>  
2593       <xsd:element ref="ImplementAction" minOccurs="0" maxOccurs="unbounded"/>  
2594       <xsd:element ref="ImplementProperty" minOccurs="0" maxOccurs="unbounded"/>  
2595       <xsd:element ref="ImplementEvent" minOccurs="0" maxOccurs="unbounded"/>  
2596     </xsd:sequence>  
2597     <xsd:attribute name="name" type="xsd:string" use="required"/>  
2598     <xsd:attribute name="title" type="xsd:string"/>  
2599     <xsd:attribute name="option" type="xsd:string"/>  
2600     <xsd:attribute name="profile" type="xsd:string"/>  
2601     <xsd:attribute name="location" type="xsd:string"/>  
2602     <xsd:attribute name="namespace" type="xsd:string"/>  
2603     <xsd:attribute name="description" type="xsd:string"/>  
2604   </xsd:complexType>  
2605 </xsd:element>
```

2606

- 2607 • *ImplementAction* element SHOULD represent an action that the program can perform for the domain
2608 document. This element MAY represent a role of the program in the transaction.
- 2609 • *ImplementProperty* element SHOULD represent a property that the program can deal with in the
2610 domain object. All properties defined in this element SHOULD be defined as a property of a domain
2611 object in the corresponding application profile.
- 2612 • *ImplementEvent* element SHOULD represent an event that the program can monitor a property in
2613 order to notify the change of the data to subscribers. This information MAY be defined by each
2614 application programs.
- 2615
- 2616 • *name* attribute SHOULD represent the name of the domain document. The name SHOULD be
2617 defined in the list of domain document in the corresponding application profile. This attribute is
2618 REQUIRED.
- 2619 • *title* attribute SHOULD represent the header title of the document. This value MAY be a short
2620 description to show the property relating to the actual world. This attribute is OPTIONAL.
- 2621 • *option* attribute SHOULD represent optional process to deal with the domain document data. There
2622 can be several domain document of same document name if the document has different option value.
2623 According to the option process, the required implement properties may be different.
- 2624 • *profile* attribute SHOULD represent the name of application profile that this *ImplementDocument* is
2625 referring to select the available part in the definition. This attribute is OPTIONAL.
- 2626 • *location* attribute SHOULD represent the location of the application profile to get the actual file by the
2627 party who want to know the content of the application profile. This attribute is OPTIONAL.
- 2628 • *namespace* attribute SHOULD represent the namespace of the *ImplementDocument*. This attribute is
2629 necessary to identify the document name in world-wide basis. This attribute is OPTIONAL.
- 2630 • *description* attribute SHOULD represent any description of the implemented document. This attribute
2631 is OPTIONAL.

2632 4.3.9 ImplementAction Element

2633 *ImplementAction* element SHOULD represent an action that the program can perform for the domain
2634 document. The actions include the transaction model referred to as “Add”, “Change”, “Remove”, “Notify”,
2635 “Sync”, “Get” or “Show”. This element MAY represent a role of the program in the transaction such as
2636 sender or receiver.

2637 This information SHOULD be specified in the following XML schema. The XML documents generated by
2638 the schema SHOULD be consistent with the following arguments.

2639

```
2640 <xsd:element name="ImplementAction">  
2641   <xsd:complexType>  
2642     <xsd:attribute name="action" type="xsd:string" use="required"/>  
2643     <xsd:attribute name="level" type="xsd:int"/>  
2644     <xsd:attribute name="role" type="xsd:string"/>  
2645     <xsd:attribute name="description" type="xsd:string"/>  
2646   </xsd:complexType>  
2647 </xsd:element>
```

2648

- 2649 • *action* attribute SHOULD represent the action performed by the application program. The value of this
2650 attribute SHOULD be one of “Add”, “Change”, “Remove”, “Notify”, “Sync”, “Get” and “Show”. This
2651 attribute is REQUIRED.
- 2652 • *level* attribute SHOULD represent an implementation level defined in Section 3 for each document
2653 processed by the application program. Level 0 shows no implementation, while level 1 and 2 are
2654 partially and fully implemented, respectively. Default value is 1 that minimum implementation is
2655 supported. This attribute is OPTIONAL.
- 2656 • *role* attribute SHOULD represent a role in the transaction. The value of this attribute is either “Server”
2657 or “Client”. Every transaction has its available roles that can be selected as a value of this attribute.
2658 Default value is “Server”. This attribute is OPTIONAL.
- 2659 • *description* attribute SHOULD represent any description of the implement action. This attribute is
2660 OPTIONAL.

2661 4.3.10 ImplementProperty Element

2662 *ImplementProperty* element SHOULD represent a domain property that can be processed in the
2663 application program. Some properties SHOULD be defined in the corresponding domain object in the
2664 application profile definition. The properties that are not defined in the application profile SHOULD be
2665 specified in this element as a user extended property. Properties extended by application programs
2666 SHOULD have additional definitions similar to the definitions by *AppProperty* element.

2667 This information SHOULD be specified in the following XML schema. The XML documents generated by
2668 the schema SHOULD be consistent with the following arguments.

2669

```
2670 <xsd:element name="ImplementProperty">  
2671   <xsd:complexType>  
2672     <xsd:attribute name="name" type="xsd:string" use="required"/>  
2673     <xsd:attribute name="title" type="xsd:string"/>  
2674     <xsd:attribute name="extend" type="xsd:string"/>  
2675     <xsd:attribute name="link" type="xsd:string"/>  
2676     <xsd:attribute name="multiple" type="xsd:string"/>  
2677     <xsd:attribute name="path" type="xsd:string"/>  
2678     <xsd:attribute name="dataType" type="xsd:string"/>  
2679     <xsd:attribute name="enumeration" type="xsd:string"/>  
2680     <xsd:attribute name="type" type="xsd:string"/>  
2681     <xsd:attribute name="use" type="xsd:string"/>  
2682     <xsd:attribute name="description" type="xsd:string"/>  
2683   </xsd:complexType>  
2684 </xsd:element>
```

2685

- 2686 • *name* attribute SHOULD represent the name of the property. The name SHOULD be defined in the
2687 corresponding application profile. This attribute is REQUIRED.
- 2688 • *title* attribute SHOULD represent the header title of the property. This value MAY be a short
2689 description to show the property relating to the actual world. This attribute is OPTIONAL.
- 2690 • *extend* attribute SHOULD represent qualifier string that is specified as prefix of the property name, if
2691 this property is extended by the local program. For example, if the value is “user”, then the description
2692 of this property will have “user:” prefix in the actual messages. This attribute is OPTIONAL.
- 2693 • *link* attribute SHOULD represent that this property is also defined in other domain document that can
2694 be linked to this document. The value of this attribute MAY has the name of domain document.
- 2695 • *multiple* attribute SHOULD represent whether the property can have multiple values or not. If the
2696 value of this attribute is positive integer or “Unbounded”, actual message can have corresponding
2697 number of values for this property. The value number SHOULD be less or equal than the number
2698 defined in the application profile.
- 2699 • *path* attribute SHOULD represent the location of the attribute data in the primitive XML description
2700 defined in Section 2. The specification of the path SHOULD conform to the syntax of [PATH]. If the
2701 attribute value of *extend* is defined and this attribute is not described, then the default path data
2702 SHOULD be “Spec[@type='aaa:bbb']/CCC/@value”, where aaa denotes the value of *extend* attribute
2703 and bbb denotes the value of *name* attribute, and CCC is the value of *dataType* attribute.
- 2704 • *dataType* attribute SHOULD represent the data type of the property. The expecting value of this
2705 attribute is Qty, Char and Time. This attribute is REQUIRED if the value of *extend* has data.
2706 Otherwise it is OPTIONAL.
- 2707 • *enumeration* attribute SHOULD represent the name of enumeration type when the property is
2708 extended by the local program, and has a value in the enumeration list. The name of enumeration
2709 type SHOULD be specified in *Enumeration* elements in the application profile definition. This attribute
2710 is OPTIONAL.
- 2711 • *type* attribute SHOULD represent that the type of this property in terms of usage. When the value is
2712 “Typical”, then the usage of this property is typical.
- 2713 • *use* attribute SHOULD whether the property is mandatory. When the value “Required” represents
2714 mandatory, while the value “Optional” represents optional. This value SHOULD be “Required” if the
2715 corresponding property in the application profile has “Required” value. Default value of this attribute is
2716 “Optional”.
- 2717 • *description* attribute SHOULD represent any description of the property. This attribute is OPTIONAL.
2718

2719 **4.3.11 ImplementEvent Element**

2720 *ImplementEvent* element SHOULD represent any event definitions that the application program monitors
2721 on a particular property and detects the event occurrence on it. When the event occurs, the application
2722 program SHOULD publish a notification of the event to all the parties who are on the list of subscription.
2723 This information is defined by each application program, then clients of the event notification service MAY
2724 request for the publication as a subscriber.

2725 *ImplementEvent* element SHOULD allow an application program to define the unit size of data
2726 differences, maximum and minimum data value, duration of one monitoring cycle and expire date of
2727 notifications to determine the event occurrence.

2728 This information SHOULD be specified in the following XML schema. The XML documents generated by
2729 the schema SHOULD be consistent with the following arguments.

2730

2731
2732
2733
2734
2735

```

<xsd:element name="ImplementEvent">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="App" minOccurs="0"/>
      <xsd:element ref="Condition" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746

```
<xsd:element ref="Selection" minOccurs="0" maxOccurs="unbounded"/>  
<xsd:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>  
</xsd:sequence>  
<xsd:attribute name="name" type="xsd:string" use="required"/>  
<xsd:attribute name="type" type="xsd:string"/>  
<xsd:attribute name="cycle" type="xsd:duration"/>  
<xsd:attribute name="start" type="xsd:dateTime"/>  
<xsd:attribute name="expire" type="xsd:dateTime"/>  
<xsd:attribute name="description" type="xsd:string"/>  
</xsd:complexType>  
</xsd:element>
```

2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787

- *App* element SHOULD represent the application specific information about event monitoring, event processing, transaction control and so forth. The specification of *App* element is defined in Section 2.
- *Condition* element SHOULD represent the condition to select the target domain objects the application is monitoring the event. The specification of this element is defined in Section 3.
- *Selection* element SHOULD represent the condition of selecting the target property in the domain object. The selected property values are reported to the subscribers when event occurs. When the target property is multiple, *Condition* element under this element can restrict the properties. The specification of this element is defined in Section 3.
- *Property* element SHOULD represent the target property and constraints to detect event on the property. The target property is monitored by the program. When there is more than one *Property* element under the *ImplementEvent*, it SHOULD represent that more than one conditions need to be checked to detect the event occurrence. Each *Property* element MAY have a different target property on the domain object to others. Conditions of these properties SHOULD be conjunctive. The specification of this element is defined in Section 3.
- *name* attribute SHOULD represent the name of the event. The name SHOULD be unique in the domain object defined in the application profile. This attribute is REQUIRED.
- *type* attribute SHOULD represent a method to detect this event. Value candidates of this attribute SHOULD include “True”, “False”, “Enter”, “Leave”, “Change”, “Add”, and “Remove”. If the value is “True”, then event occurs when all the conditions are true. If the value is “False”, then event occurs when at least one condition is false. If the value is “Enter”, then event occurs when the status changes from false to true, while “Leave” means that the status changes from true to false. If the value is “Change”, then event occurs when the value of the target property is change. “Add” represents that event occurs when a new domain object which satisfies the conditions is added, and “Remove” shows that event occurs when any objects which satisfies the conditions is removed. If the target property is multiple and *Selection* element is described, then “Add” and “Remove” mean that one of the multiple properties is added and removed, respectively. Default value is “Change”. This attribute is OPTIONAL.
- *cycle* attribute SHOULD represent the duration of monitoring of the property value to detect the event occurrence. The application program SHOULD monitor the value until the expiration date. This attribute is OPTIONAL.
- *start* attribute SHOULD represent starting time of the monitoring and notification service. After this date and time, application program start monitoring the properties. If this attribute is not described, then it represent the service has already started. The origin of cyclic procedure defined by *cycle* attribute SHOULD be this start time. This attribute is OPTIONAL.
- *expire* attribute SHOULD represent expire time and date of the event notification. After the time of expiration, the application will stop monitoring the event occurrence. If this attribute is not defined, it SHOULD represent that there is no expiration date. This attribute is OPTIONAL.
- *description* attribute SHOULD represent any description of the event. This attribute is OPTIONAL.

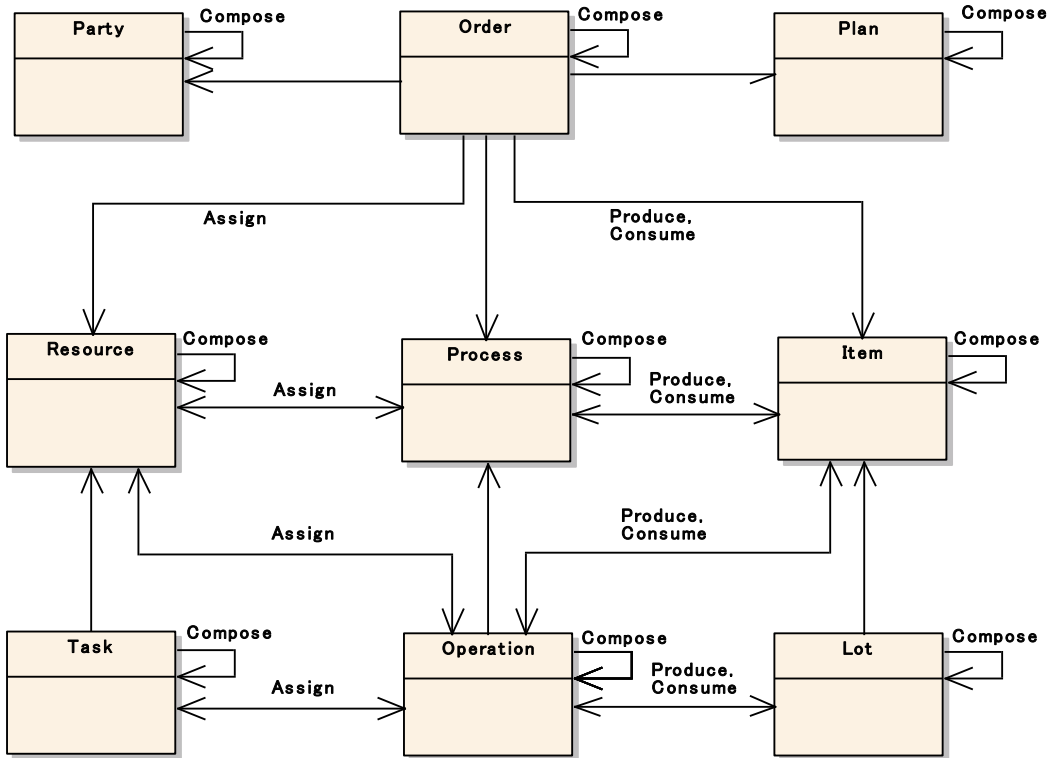
2788 **5 Conformance**

- 2789 A document or part of document conforms to OASIS PPS Core Elements if all elements in the artifact are
2790 consistent with the normative statements of Section 2 of this specification and the document can be
2791 processed properly with the XML schema that can be downloaded from the schema URI.
- 2792 A document or message conforms to OASIS PPS Transaction Messages if all elements in the artifact are
2793 consistent with the normative statement of Section 3 of this specification and the document can be
2794 processed properly with the XML schema that can be downloaded from the schema URI.
- 2795 A process or service conforms to OASIS PPS Transaction Messages if the process or service can deal
2796 with the message that conforms to OASIS PPS Transaction Messages and the process or service is
2797 consistent to the normative statement of Section 4 of this specification.
- 2798 A document or profile conforms to OASIS PPS Profile Specifications if all elements in the artifact are
2799 consistent with the normative statements of this part of specifications and the document can be
2800 processed properly with the XML schema that can be downloaded from the schema URI.
- 2801 The schema URI is given in the “Related work” section in the header page of this document.

2802

Appendix A. Object Class diagram of Core Elements

2803 Figure A.1 shows the structure of primitive objects in this specification with a UML class diagram. Each
2804 object corresponds to each XML element. In this figure, arrows represent relative information between the
2805 source and destination objects. When an arrow has role names, it corresponds to an independent XML
2806 element in the specification. This figure doesn't include all the information of XML schema but the partial
2807 information of the primitive elements.
2808



2809

2810 Figure A.1: Primitive objects for representing planning and scheduling problems

2811

2812

Appendix B. Cross reference of elements

2813 Table B.1 shows the relations between elements. The row headers represent parent elements and the
 2814 column headers represent child elements. Symbol * in the table means 0 or more than 0 element can be
 2815 described.

2816

2817 Table B.1 Element and sub-element relations

	Compose	Produce	Consume	Assign	Relation	Location	Capacity	Progress	Spec	Start	End	Event	Price	Cost	Priority	Display	Description	Author	Date	Qty	Char	Time
Party	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Plan	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Order	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Item	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Resource	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Process	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Lot	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Task	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Operation	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			
Compose						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Produce						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Consume						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Assign						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Relation						*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Location										*	*	*	*	*	*	*	*	*	*	*	*	*
Capacity										*	*	*	*	*	*	*	*	*	*	*	*	*
Progress										*	*	*	*	*	*	*	*	*	*	*	*	*
Spec										*	*	*	*	*	*	*	*	*	*	*	*	*
Start															*	*	*	*	*	*	*	*
End															*	*	*	*	*	*	*	*
Event															*	*	*	*	*	*	*	*
Price															*	*	*	*	*	*	*	*
Cost															*	*	*	*	*	*	*	*
Priority																				*	*	*
Display																				*	*	*
Description																				*	*	*
Author																				*	*	*
Date																				*	*	*
Qty																						
Char																						
Time																						

2818

2819

2820

2821

2822 The following table B.2 shows the correspondence between elements and attributes. The row headers
 2823 show the element name, and the column headers show attribute the name. The characters in the table
 2824 represent data types. The character in the table are used as follows: "U" denotes identification character
 2825 of element, "P" denotes an identification character of referencing elements, "S" denotes the character
 2826 string, "D" denotes a decimal number, "N" denotes an integer number and "T" for date time. Boldface
 2827 means required information.

2828

2829 Table B.2 Element and attribute relations

	id	key	name	parent	type	status	apply	condition	value	count	unit	base	party	plan	order	item	resource	process	lot	task	operation
Party	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Plan	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Order	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Item	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Resource	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Process	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Lot	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Task	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Operation	U	N	S	P	S	S							P	P	P	P	P	P	P	P	P
Compose	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Produce	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Consume	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Assign	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Relation	U	N	S		S	S	S						P	P	P	P	P	P	P	P	P
Location	U	N	S		S	S	S														
Capacity	U	N	S		S	S	S														
Progress	U	N	S		S	S	S														
Spec	U	N	S		S	S	S														
Start	U	N	S		S	S	S	S	S												
End	U	N	S		S	S	S	S	S												
Event	U	N	S		S	S	S	S	S												
Price	U	N	S		S	S	S	S	S												
Cost	U	N	S		S	S	S	S	S												
Priority			S		S	S	S	S	S												
Display			S		S	S	S	S	S												
Description			S		S	S	S	S	S												
Author			S		S	S	S	S	S												
Date			S		S	S	S	S	S												
Qty			S		S	S	S	S	D	N	S	D									
Char			S		S	S	S	S	S	N	S	S									
Time			S		S	S	S	S	T	N	S	T									

2830

2831 **Appendix C. Implementation level**

2832 Since this specification provides the highest level functionality of application programs of information
2833 exchange on planning and scheduling problems, it might be difficult to implement for the application
2834 programs that don't need full capability of messaging. Regarding such situation, this specification
2835 additionally defines implementation levels for each application program.

2836 The implementation level is specified in implementation profiles defined in Section 4. Each application
2837 program MAY describe its capability for each messaging model. Therefore, system designer of the
2838 domain problem can know available combination of messaging without making a configuration tests.

2839 The following table prescribes the implementation levels.

2840

2841 *Table C.1 Implementation levels*

Level	Description
0	The application program has no capability of the function
1	The application program has some capability of the function. The partial function is defined for the restricted specifications.
2	The application program has all capability on the function prescribed in this standard

2842

2843 There are some functional categories of specifications, in which some additional constraints MAY be add
2844 to restrict the full specification. The level 1 of implementation is conformed to this restricted specification.
2845 In this specification, "Level 2 Function" denotes that the section or subsection is not necessary for the
2846 application program that declares level 1 for the messaging model.

2847

Appendix D. Revision History

2848

Revision	Date	Editor	Changes Made
01	23 Feb 2011	Y.Nishioka	Marge three parts of CS01
02	24 May 2011	Y.Nishioka	Name space URI and Cover page URI are revised

2849

2850

Appendix E. Acknowledgements

2851 The following individuals have participated in the creation of this specification and are gratefully
2852 acknowledged:

2853 **Participants:**

2854 Yasuyuki Nishioka, PSLX Forum/Hosei University

2855 Koichi Wada, PSLX Forum

2856 Shinya Matsukawa, Hitachi

2857 Tomohiko Maeda, Fujitsu

2858 Masahiro Mizutani, Unisys Corporation

2859 Akihiro Kawachi, Individual Member

2860 Yuto Banba, PSLX Forum

2861 Osamu Sugi, PSLX Forum

2862 Hideichi Okamune, PSLX Forum

2863 Hiroshi Kojima, PSLX Forum

2864 Ken Nakayama, Hitachi

2865 Yukio Hamaguchi, Hitachi

2866 Tomoichi Sato, Individual

2867 Hiroaki Sasaki, Individual

2868 Tomoichi Sato, Individual

2869 Junzo Kato, PSLX Forum

2870 Hiroaki Machida, PSLX Forum

2871 Shoei Komatsu, PSLX Forum

2872

2873